AFRL-IF-RS-TR-2006-182
Final Technical Report
May 2006

AUTONOMIC FUSELET SPECIFICATION AND COMPOSITION

ORIELLE, LLC

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2006-182 has been reviewed and is approved for publication

APPROVED: /s/

DALE W. RICHARDS
Project Engineer

FOR THE DIRECTOR: /s/

JAMES W. CUSACK
Chief, Information Systems Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
MAY 2006	FINAL	Mar 05 – Mar 06

4. TITLE AND SUBTITLE

AUTONOMIC FUSELET SPECIFICATION AND COMPOSITION

5a. CONTRACT NUMBER
FA8750-05-C-0088

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62702F

6. AUTHOR(S)

Peter H. Mills

5d. PROJECT NUMBER
558J

5e. TASK NUMBER
TD

5f. WORK UNIT NUMBER
04

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
ORIELLE, LLC
121 West Sweet Ave, Suite 124
PO Box 8922
Moscow ID 83843

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFSE
525 Brooks Rd
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2006-182

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; distribution unlimited. PA# 06-370

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A framework for autonomic fuselet business logic development was developed, using semantic web services and workflow technologies to specify fuselet information needs, to define an executable workflow model for its business logic, and to perform workflow composition from a goal and specifications to executable model in a manner that enables feedback mechanisms that can assess and adapt the process model to satisfy needs. The framework for autonomic fuselet specification and composition consists of: (a) the definition of ontology-based fuselet specifications in a variant of the Web Ontology Language for Services (OWL-S), (b) the definition of fuselet business logic in the Business Process Execution Language (BPEL), and (c) a workflow composition engine that composes a fuselet-based workflow from other fuselets by deriving a BPEL process model from a fuselet goal and a set of fuselet specifications.

15. SUBJECT TERMS

Joint Battlespace Infosphere (JBI), Fuselets, Semantic Web, Web Services, Workflow, Ontology, OWL, BPEL

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dale W. Richards
U	U	U	UL	101	19b. TELEPONE NUMBER (*Include area code*)

Table of Contents

List of Figures

Summary

A fuselet is typically a persistent process that communicates only by publish, subscribe, and query in order to perform information transformation. It is intended to be deployed, for example, within the Joint Battlespace Infosphere, a framework for global information sharing and information management. Fundamental to the viable use of fuselets is technology that enables the succinct expression of fuselets as well as their management in a runtime framework that supports autonomic execution, that is, assessment and reengineering of the fuselet to meet information needs. Key in turn to such autonomic computation is a method for reengineering fuselets: one such method is that of automated workflow composition.

The overall objective of this research effort was to develop a proof-of-concept demonstration of autonomic fuselet business logic development that uses semantic web services and workflow technologies to specify fuselet information needs, to define an executable workflow model for its business logic, and to perform workflow composition from a goal and specifications to executable model in a manner that enables feedback mechanisms that can assess and adapt the process model to satisfy needs. The specific objectives were to: (a) define an ontology-based fuselet specification expressed in a variant of the Web Ontology Language for Services (OWL-S), (b) define a notation for fuselet business logic expressed in a restricted subset of the Business Process Execution Language (BPEL), together with the identification of a set of basic manipulation functions to flesh out the business logic, and (c) design and implement a workflow composition engine for deriving a BPEL workflow model from a goal and a set of fuselet specifications.

This report first describes the notation and standards used for ontology-based fuselet needs specification. Then specification of a fuselet's information needs is considered, that is, what it consumes and produces, using emerging semantic web service standards. Specifically the behavior of a fuselet is captured in OWL-S using Prolog-like input and output relations in a form that is amenable to reasoning using conventional theorem provers enhanced with ontology subtyping.

Techniques for capturing a fuselet's business logic, that is, the internals of how it consumes and produces information to satisfy its specified needs, in an executable workflow model using Web Services and the Business Process Execution Language (BPEL) are then described. A key facet underlying this approach is the modeling of fuselets as web services. Expressing a fuselet as a workflow involves both capturing its interface using the Web Service Definition Language (WSDL) and its executable logic using BPEL. It is also advantageous to expose the publish and subscribe operations invoked by the fuselet as a BPEL process. The use of workflow technologies such as BPEL to implement fuselets enables both their assembly at the micro-scale using Joint Battlespace Infosphere (JBI) Common API (CAPI) publish-subscribe operations, as well as the composition of fuselets at the macro-scale to realize collective behavior, for example, in demand-driven workflow composition.

Lastly, the design and implementation of a workflow composition engine that composes fuselets from other fuselets by deriving a BPEL workflow model from a fuselet goal and other fuselet

specifications is described. In the design of the workflow composition engine a broad spectrum of candidate planning technologies were evaluated with an eye towards those that could have their inference mechanisms extended to support ontology-based sub-typing within a Java environment.

Such a workflow composition engine can be used at the both the micro-scale in a fuselet production environment, that is, to develop business logic for a single fuselet and semi-automatically guide its development by choosing from palette of subtyped manipulation functions and semantically matching fuselets, and at the macro-scale for demand-driven adaptive workflow composition. In particular within the larger picture of autonomic computing, such a workflow composition engine is meant to work in tandem with monitoring and feedback mechanisms that assess the fuselet output behavior, that is, compares its outputs to the original specification, and then re-engineer the workflow model to improve and maximize the value of information produced by the fuselet. As part of this research effort metrics and feedback mechanisms were examined as to their potential for enabling assessment and re-engineering.

The report concludes a description of lessons learned and recommendations for future work, followed by an appendix listing example specifications and demonstration programs.

1. Problem Statement

A next step in the evolution of complex large-scale systems is that of autonomic computing, which like the human nervous system is self-managing, self-configuring, self-healing, and self-protecting without conscious management. An autonomic computing system instruments the underlying computing elements with sensors and effectors, and then uses independent agents to monitor, assess, plan, and execute adaptations to the elements and their configuration in order to achieve goals based on policies and metrics. As is becoming evident from a consensus in many research efforts, the building blocks of such adaptive survivable systems are a confluence of semantic web services [OWL-S], grid computing, workflow-based business process logic specifications and execution engines [BPEL-spec], and automated program construction techniques based on theorem proving and planning that can effect both workflow adaptation on the large scale and, on the micro-scale, semi-automated program synthesis from specifications [ACC04, MWG04, RKM04]. Building such systems requires a development environment that from the ground up incorporates the required concepts of logical specification and semi-automatically derivable process logic that can be monitored to see if it satisfies behavior metrics. Not surprisingly, and as alluded to above, the same techniques and machinery that effect adaptation through monitoring and planning can be applied to guide a developer from specification to program assembly.

In particular, the above building blocks are key to a fuselet production environment whose artifacts are to be used in an autonomic fuselet runtime system that can monitor and adapt collective fuselet behavior to meet end-user information requirements. A fuselet is typically a persistent lightweight process that communicates only by the medium of publish-subscribe, in particular using emerging global information frameworks such as the JBI that support such a communication paradigm. A fuselet's business logic can be couched in notations ranging from Java-based scripting languages to web service composition languages such as the Business Process Execution Language (BPEL), recently ratified as an Organization for the Advancement of Structured Information Standards (OASIS) standard. In particular, BPEL is a means of invoking and composing other web services in a workflow orchestration, and is expressed in the Extensible Markup Language (XML) itself. BPEL is particularly attractive as a means of specifying a fuselet's business logic, since it provides simple constructs for XPath-based selection of inputs, akin to those used for subscription message selectors, as well as a comprehensive array of constructs for orchestrating programs including guarded nondeterministic message-passing, used to form the so-called "process logic". While it doesn't directly provide for low-level manipulation and computation functions, BPEL has placeholders for augmenting its process logic with functions sufficient to yield the "business logic" of fuselets.

From an autonomic perspective, fuselet development and execution entails more than just providing its business logic. Fuselet business logic development and execution proceeds from: (a) a high-level specification of its needs, through (b) construction of its business logic using other fuselet components to yield a fuselet workflow model that satisfies the needs, to (c) monitoring its execution in a way that compares its outputs to the original specification, and using these feedback mechanisms to modify and reengineer the original workflow. Key in turn

to such autonomic computation is a method for reengineering fuselets: one such method is that of automated workflow composition.

One emerging technology for needs specification that fits into this autonomic composition framework is that of semantic web services, specifically the Web Ontology Language for Services (OWL-S), that describes process behavior using an ontology-based XML specification that provides a service profile to define its surface logic of its behavior, a process model to describe its internal execution, and a grounding to map the model onto web services standards such as WSDL for invocation and Universal Description, Discovery and Integration (UDDI) for discovery. Ontologies define relations, properties, and rules of inference about terms used in XML documents. Concomitantly with OWL-S there are several research efforts which have used it as a driving vehicle for automatic workflow composition, specifically derivation of BPEL or its equivalent for the process model [ACC04, MWG04, and RKM04]. A promising approach to fuselet business logic development and its execution in an autonomic environment is thus to leverage these emerging technologies for semantic web services and workflow orchestration.

2. Objective

The objective of this research effort was to develop a proof-of-concept demonstration of autonomic fuselet business logic development that uses semantic web services and workflow technologies to specify fuselet information needs, to define an executable process model for its business logic, and to perform workflow composition from a goal and specifications to executable model in a manner that enables feedback mechanisms that can assess and adapt the process model to satisfy needs. Specifically this research effort sought to develop an ontology-based specification, workflow process model, and workflow composition engine for fuselets that adapts emerging technologies such as the Web Ontology Language for Services (OWL-S) and BPEL (Business Process Execution Language) to the domain-specific problem of autonomic fuselet composition and execution.

It is imperative to adopt or adapt emerging technologies such as OWL-S for semantic web services and BPEL for composing web services as building blocks for a fuselet production environment, both to track technology and to exploit techniques such as OWL/BPEL-based automated workflow composition algorithms as well as tools such as graphical editors that are part and parcel of BPEL implementations. A fundamental observation is that it may be possible to significantly enhance the viability and robustness of general techniques in semantic web services and automated workflow composition by restricting the domain of discourse to the fundamental characterization of fuselets, i.e., that they can only interact through publish, subscribe, and query operations.

This research effort can serve as a foundation for the specification and guided synthesis of fuselets in a manner that can be readily expanded into advanced semantics-based fuselet authoring and autonomic runtime management capabilities.

3. Technical Approach

The technical approach to realizing a foundation for autonomic fuselet specification and composition consists in developing: (1) An ontology-based fuselet specification expressed in a variant of OWL-S, which uses the service profile to capture both a Horn Clause (Prolog-like) logic specification with basic manipulation terms and message-selector types taken from an ontology, as well as potentially augmented with Quality of Service (QOS) metrics that map process input and output behavior onto ontology-based satisfaction terms, (2) a notation for fuselet business logic expressed in BPEL to capture the process model, together with basic manipulation functions to flesh out the business logic, and (3) a workflow composition engine for deriving a BPEL workflow model from a goal and a set of fuselet specifications, that can operate both at the micro- and macro-scale.

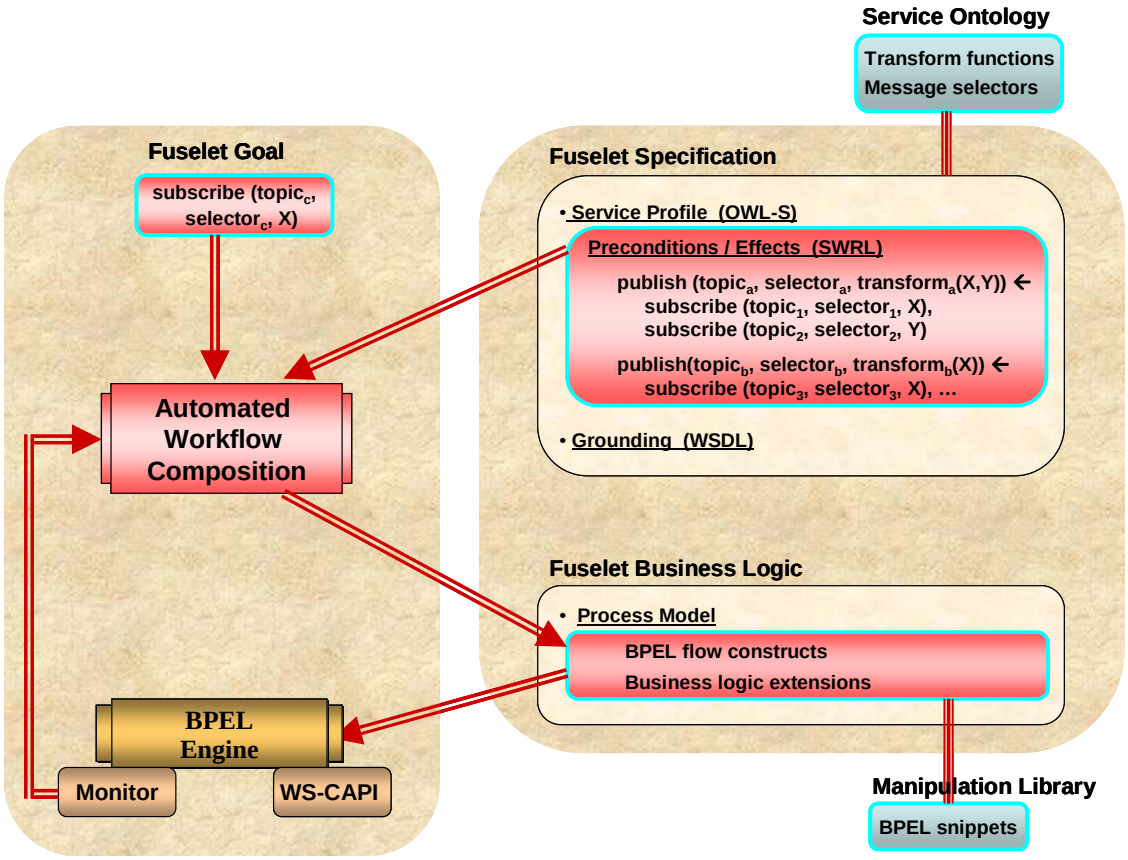

Figure 1. Architecture for Autonomic Fuselet Specification and Composition

Figure 1 illustrates the architecture for autonomic fuselet specification and composition, and how it fits in with the larger vision of an autonomic fuselet runtime environment. In Figure 1, the upper right in red area shows the fuselet service specification using a variant of OWL-S preconditions and postconditions to capture the relations between outputs (“publish” on the left-hand side of an if-statement) and inputs ("subscribe" on the right-hand side) that maximally

cover all possible behaviors. Within the logic specification, a hierarchy of terms denoting message-selector types and manipulation functions, shown in the upper right hand of the figure, are used. Ontology-based specification of selectors and manipulation functions allows leveraging ontology-based browsers to guide novice uses in indicating the type of information required, and is also key to enabling ontology-based inference that extends resolution with ontology subtyping. Also in the service profile may be a Quality of Service (QOS) metric (not shown in the figure) that specifies, for example, how to map actual XPath-based selectors onto selector terms.

The business logic of the fuselet is described using BPEL together with a referenced set of manipulation functions to flesh out the business logic, shown in the lower right of Figure 1. Strictly speaking, an OWL-S specification consists of a service profile, a grounding that maps onto concrete substrates such as WSDL, and a process model typically expressed in a situation calculus such as ConGolog that is used as a plan in Hierarchical Task Network (HTN) planners. In the domain of fuselets, BPEL suffices as a more concrete description of the process model.

The business logic is then run on a conventional BPEL engine, shown in the lower left. The idea in autonomic computing is that monitors attached to the engine or instrumented into the code can capture the runtime behavior, and analyze it to see if it meets the specified user needs. If not, or to maximize the value of the information, the workflow can be reengineered. When the process model captures a collection of fuselets, reengineering in that case reduces to workflow composition from a goal and specifications to a BPEL workflow. This workflow is then run on the engine, to complete the analyze-plan-execute cycle.

The mechanics of workflow composition have dual-use both at the macro-scale as above to dynamically adjust collective fuselet behavior to satisfy metric-based policies, as well as at the micro-scale in a fuselet production environment, that is, to develop business logic for a single fuselet and semi-automatically guide its development by choosing from palette of subtyped manipulation functions and semantically matching fuselets.

This research and development effort was divided into three tasks that address the above technical objectives. These tasks were:

Task 1. Define Ontology-Based Fuselet Logic Specification.

In this task an ontology-based fuselet logic specification was expressed in a variant of OWL-S, which uses the service profile to capture a Horn Clause (Prolog-like) logic specification with basic manipulation terms and message-selector types taken from an ontology. The Horn clauses are basically if-then clauses of the form:

```
publish(topic, outCondition, transform(X,Y))

      IF  subscribe(topic1,  selector1,  X)    AND    subscribe(topic,2,
selector2, Y)
```

The logic is enhanced to make the selectors terms in an ontology, and to use ontology-based subtyping in its inference of what matches the arguments during Prolog-like resolution. The above approach separates the problem of deciding which fuselet inputs match other fuselet

outputs into two stages: a first stage that maps XPath-based input selectors into ontology terms, and a second stage that uses those terms to reason about fuselet composition and needs satisfaction.

Task 2. Define BPEL-Based Fuselet Business Logic.

In this task capture the fuselet process model in BPEL together with manipulation functions to flesh out the business logic were examined. Asynchronous techniques to invoke the publish-subscribe API exposed as a BEPL web service were also examined.

Task 3. Design and Implement a Workflow Composition Engine.

In this task a workflow composition engine that can derive a BPEL workflow model from a goal and a set of fuselet specifications was developed. Its algorithm extends Prolog-like inference on the specifications with ontology-based subtyping. The implementation leverages the Prova/Manadarax inference engine and the Jena Semantic Web Toolkit.

Use of metrics that map XPath-selectors characterizing published results to their equivalent ontology terms in the specification: such metrics might be used to incorporate metric-based feedback mechanisms into the larger autonomic cycle which would then re-engineer the workflow orchestration to optimize needs satisfaction was also considered and documented.

The remainder of this document describes the key results from the research effort, and concludes with a summary of lessons learned and recommendations for further research. Lastly, concrete examples of fuselet specification and composition in a demonstration scenario of radar tracking from sensor returns are provided.

4. Ontology-Based Fuselet Specification

4.1 OWL-S for Specification of Fuselet Information Needs.

4.1.1 What is OWL-S

The ontology-based fuselet logic specification is expressed in a variant of the Web Ontology Language for Services (OWL-S). OWL-S uses OWL, the successor to DAML and a draft W3C standard, to provide an upper ontology model for web services. OWL, in turn, is a notation for describing ontologies in the Extensible Markup Language (XML), where ontologies specify the hierarchy and properties of terms used in documents. An OWL-S specification for a process has three sections:

(1) A service profile, which specifies what the service provides,

(2) A process model, which specifies how the service does it, and

(3) A grounding, which specifies how to access the service, i.e., the mapping to concrete web service specifications such as WSDL.

4.1.2 Benefits of OWL-S

It is important to examine the technical feasibility of using OWL-S as a fuselet specification for several reasons. First, OWL-S is a recent draft W3C standard, so it may possibly be widely adopted. Second, OWL-S is based on OWL which in turn is based on the Resource Description Framework (RDF) and RDF-Schema, the fundamental enabling technologies for the "semantic web" which uses XML to express subject-verb-object triples. Using such a semantic-based web service specification enables using the array of emerging web technologies for semantics-based process discovery, enactment, and composition. Third, OWL-S is largely driven by AI planning notations, and incorporates in its service profile more than just inputs and outputs, but also preconditions and effects, which can be used for AI planning and web service composition. The latter has in particular potential advantages for specifying fuselets within an autonomic or self-configuring runtime environment, where here autonomic means the ability to assess whether information needs are being satisfied by the fuselet execution, and as needed to re-engineer the fuselet or workflow of fuselets.

4.2 Applying OWL-S to Fuselets

4.2.1 Service profile, process model, and grounding

For the purposes of this effort, and driven in part by an evaluation of how workflow composition using AI planning techniques would be used for fuselets, OWL-S was applied to fuselet specification as follows. First, as is described in more detail below, a portion of the service profile's Parameters and Inputs/Outputs/Preconditions/Effects (IOPE) were used to capture a Horn Clause ("Prolog-like") logical specification for fuselet publish and subscribe actions with basic manipulation terms and message-selector types taken from an ontology. The if-then Horn

clauses are coded using the Semantic Web Rule Language (SWRL) which is admissible under OWL-S.

Second, the usual composite process model for OWL-S, which might typically be encoded using SWRL or a situation calculus such as ConGolog, was largely bypassed and an executable process definition in the Business Process Execution Language (BPEL) substitute for it. OWL-S provides three types of process models: atomic, which corresponds to directly invocable web services, composite, which corresponds to using flow constructs invoking other services to build up a service, and simple, which corresponds to an abstract view of a composite process.

A simple abstract process view of the fuselet (which actually holds the parameter and IOPE definitions that are in turn referenced from the profile), as well as a composite process definition to hold the BPEL are provided. Since Hierarchical Task Network (HTN) planning for web service composition, which uses the composite process model as a plan template to derive an executable workflow, will not be used, an alternative executable model for the OWL-S composite flow constructs can be substituted.

Third, a form of grounding is encoded in the OWL-S for fuselets, to enable the execution of web service composition results using BPEL and WSDL.

It bears emphasizing that using a notation such as OWL-S does not require *a priori* that the fuselets be web services, although there is significant merit in modeling fuselets as web services. Rather, as mentioned earlier, using semantic web notations enables the application of ontology-based inference techniques, and provides a richer description of the relations between fuselet inputs and outputs that includes a more logic-like specification of preconditions and effects.

4.2.2 Encoding fuselet information needs using if-then clauses

The basic approach taken in this effort was to increase the robustness of reasoning by restricting the domain of discourse to the fundamental characteristics of fuselets, specifically to restrict an otherwise general OWL-S specification to a simple logic formulation whose terms are publish-subscribe operations.

Parameterized fuselets are specified using sets of if-then statements, with one if-then statement for each type of published information, of the form:

```
tracker_fuselet(Z) {
        publish(topicOut1, selectorOut1, transformOut1(X,Y,Z))
                IF subscribe(topicIn1, selectorIn1, X)
                AND subscribe(topicIn2,  selectorIn2, Y)
        publish(topicOut2, selectorOut2, transformOut2(X,Y,Z))
                IF subscribe(topicIn3, selectorIn3, X)
                AND subscribe(topicIn4, selectorIn4, Y)
        [...]
}
```

where:

`topics`: are JBI information object types,

`selectors`: are JBI selector predicates characterizing input or output conditions,

`transforms`: are transform functions,
`X, Y, Z`: are logical variables.

The above describes which inputs must be provided to produce each output type, and is basically a Horn Clause (Prolog-like) logic specification with manipulation terms and message-selector types taken from an ontology. As in Prolog, discovery or AI planning is used to determine what other fuselets must be run to provide the required subscriptions for this fuselet, term matching (also called unification) is used to decide which fuselet's publications match or refine the required subscriptions. Unlike Prolog, here term matching on transforms and selectors uses ontology subtyping. This means that if another fuselet publishes a more refined type of information than what is required for this fuselet, running it would still provide what is needed.

In general, the minimal requirement for deciding whether a subscription demand can be satisfied by another fuselet's publication is that the selectors intersect or overlap; that is, a subscription is admissible as a match if its concrete XPath selector intersects with that of a publication. For purposes of this effort, and in line with some of the workflow composition literature [ACC04, MWG04], ontology subtyping is used as the criteria for matching. Such subtyping translates directly into SWRL as OWL subclass predicates. However, as is discussed in the conclusion, more complex OWL and SWRL statements could be used to capture more sophisticated matching criteria.

For example, the specifications for a threat fuselet that as a consumer depends on the publications of a producer tracker fuselet are as follows:

Threat Fuselet:
```
publish( threatTopic, threatSelector, threatTransform (X))
IF subscribe( trackTopic, trackSelector,  X).
[…]
```

Tracker Fuselet:
```
publish( trackTopic, refinedTrackSelector, trackTransform
(X, Y))
IF subscribe( sensorTopic,    sensorSelector,   X)  AND
subscribe( sensorTopic2,  sensorSelector2, Y).
```

Here, reasoning uses ontology subtyping on selector terms to infer that the demand for a subscription is satisfied by another fuselet publishing a refined type, in this case that the trackSelector subscription of the threat fuselet can be satisfied by the refinedTrackSelector publication of the tracker fuselet. A separate stage would map from predicates (XPath) to selector terms, which might also be captured in the service profile; such a mapping would then be used as a metric for feedback mechanisms that compare the output to the needs as an actuator for reengineering. It also bears noting that, for the initial formulation, query is treated identically to subscribe, since a fuselet that needs a query would require a publication to have occurred in the past to satisfy it; a more refined approach might be along the lines of temporal logic that distinguishes past occurrences from those ongoing.

Such a notation enhances simpler versions of fuselet metadata that might only specify inputs and outputs, by additionally providing preconditions and effects. Such a Prolog-like specification can then be used by theorem provers such as Prova, with its inference mechanisms enhanced to

use ontology-based subtyping when unifying terms during resolution, to perform automated fuselet composition. The above notation has the advantage of separating the decision as to which fuselet inputs match other fuselet outputs into two stages: a first stage that maps XPath-based input selectors into ontology terms (which can be recorded in the ontology or in the fuselet specification as a nonfunctional attribute), and a second stage that performs ontology-aware reasoning.

4.2.3 Casting the if-then clauses into OWL-S

The above logic specification can be recast into OWL-S in several ways; one approach is to treat subscribe and publish operations as preconditions and effects, and set the inputs and outputs to be the maximal covering of the publish and subscribe topics.

In particular, the OWL-S service profile supports specification of process capabilities roughly of the form:

```
process(parameters) {
        if precondition then
                take inputs ;
                foreach (result) {
                        if inCondition then { hasEffects ; withOutputs }
}
```

where the inputs and outputs represent information-gathering, and preconditions and effects represent stateful world-altering events. The conditions and effects can be represented in a notation such as SWRL, an extension of RuleML -- which is basically Horn logic or if-then statements – with OWL binary and unary predicates.

OWL-S has a slight weakness in that it does not have a concept of "withInputs"; in other words, it assumes a constant set of inputs are always used for different results and their conditions and effects. OWL-S also partitions the preconditions and effects for each result into separate XML sections, which precludes the uniform use of SWRL to express the simple if-then statements. The former is addressed by restricting the above general form of the OWL-S service profile to:

```
process(parameters) {
        foreach (result) {
                if inCondition then { hasEffects ; withOutputs }
        }
}
```

For a fuselet specification, there is one OWL-S result for each publish if-then clause, where the "IF" subscribe terms form the "inCondition", and the left-hand side publish terms form the "hasEffects".

To accommodate SWRL restrictions, each of the publish if-then clauses is internally represented using binary predicates and unary class/data-range predicates, yielding the following in OWL-S:

```
Parameter: Z is_of_type String
Input:  X  is_of_type topicIn1
Input:  Y  is_of_type topicIn2
Output: T1 is_of_type topicOut1
hasResult {
        incondition {
                selectorIn1(X) AND selectorIn2(Y)
        }
        hasEffect {
                selectorOut1(T1) AND transformOut1(T1, [X,Y,Z])
        }
}
hasResult ... // for each publish if-then clause
```

where:

`topics` are subclasses of JBIInfoObject, and
selectors are classes arranged in a hierarchy.

4.3 Alternatives

4.3.1 Using SWRL or PDDL to capture needs

To address the second drawback mentioned above concerning the fragmentation of the succinct Horn clauses into separate preconditions/effects in OWL-S, one could bypass the OWL-S service profile entirely and express the Horn clauses directly in SWRL. A further step would be to use an XML version of a planning notation such as the Planning Domain Definition Language (PDDL) for STRIPS-like classical planning (or DRS which is an XML-based equivalent), which would adequately capture each fuselet publish-behavior as a parameterized operator with preconditions (subscribes) and effects (publications). However, to keep in line with emerging standards, OWL-S will be used, although it is a bit bulky.

4.3.2 Guards

Along the lines of Concurrent Prolog or Guarded Horn Clauses, one could add guards to the fuselet specification, of the form:

```
publish(topic, selector, transform(X,Y,Z))
        Guard: IF subscribe(topic1, selector1(X))
        AND subscribe(topic2, selector2(Y))
```

The interpretation of such a guard in concurrent logic languages is that it specifies a constraint to be satisfied before committing to that choice of the if-then clause (so-called committed choice nondeterminism used in evaluating concurrent logic languages). In this case one should interpret such a guard as a condition under which outputs would be produced given that the appropriate inputs were received.

Such a specification closely mirrors a very simple high-level specification of atomic fuselet actions of the form:

`Input:`	info-object type
`Input predicate:`	XPath expression
`Output:`	info-object type
`Operator:`	transforms outputs from inputs
`Condition:`	condition on inputs for outputs to fire

Guards thus might permit direct generation of fuselet code from the specification for simple fuselets. OWL-S and SWRL can easily express such guards as an additional conjunctive in the precondition. In the initial prototype demonstration guards were omitted, with the recognition that semantic web service specifications can accommodate them.

4.4 Metrics and Ontology Derivation

To eventually enable the full cycle of autonomic fuselet execution, one must be able to perform both metric-based fuselet assessment as well as web service composition in response to assessment. For the former, metrics that map XPath-based selectors to ontology terms must be encoded in the fuselet specification. For practical derivation of fuselet specifications, one also might consider how to automatically infer ontology subtyping of terms from their XPath selectors.

Both metric-based assessment and automatic inference of ontology subtyping from XPath selectors is beyond the scope of the research effort. The focus of the current research effort is on specification and web service composition, which is an enabling step for the larger problem of autonomic execution; metric-based assessment as well as automatic selector ontology derivation is left for future research.

4.5 Example Specification

An example of how a simple fuselet specification would appear in OWL-S is provided in Appendix A for the previously described tracker fuselet. Also provided is a sample ontology of selector terms.

5. BPEL-Based Fuselet Business Logic

The goal of this task is to examine techniques for expressing fuselet business logic in the Web Services Business Process Execution Language (BPEL), both at the level of individual fuselets as well as for fuselet collections.

The approach taken for capturing individual fuselets as BPEL workflows has three distinguishing features:

- Using Java snippets for manipulation functions.
- Encapsulating the JBI CAPI in BPEL.
- Providing a generic fuselet template mapping an input topic and selector through a transform to an output topic and condition, with exposed control points for start, stop, pause, and resume.

For capturing fuselet collections as BPEL workflows, used in workflow composition, the technique:

- Expressed composition as concurrent invocations (flow construct).
- Used correlation sets to manage fuselet instances.

The motivation and techniques for expressing fuselets in BPEL are described in detail below.

5.1 BPEL for Capturing Fuselet Business Logic.

5.1.1 What is BPEL

The convergence of workflow specifications into the Web Services Business Process Execution Language (BPEL) and the concomitant emerging workflow technologies including those of Oracle, IBM, and Active Endpoints, have made real the practical and standards-based use of workflow orchestration of XML-based web services. At its heart, BPEL provides high-level parallel-programming constructs for coordinating groups of potentially long-lived stateful web service sessions, using correlation sets based on message content to route to specific stateful web service instances. A BPEL specification of a workflow consists of three parts:

- Web Services Definition Language (WSDL) specifying the interfaces, including additional specifications for partner links that map roles to WSDL operations.
- BPEL source code, which consists of an XML-based language that presents a message-passing calculus for the process model, and includes constructs for:
    - Partner Links: map roles to WSDL operations
    - Correlation Sets: which fields identify stateful web service instances
    - Variables and properties
    - Atomic operations (receive, reply, invoke, assign)

    - Flow constructs: concurrency with Petri-net-like synchronization
    - Fault Handling
    - Compensation Handling (rollback)
- Java extensions (snippets) for manipulation functions.

The BPEL workflow together with the manipulation functions yields what is termed the business logic. The BPEL source code may be compiled and executed on any one of several vendor or public-domain substrates. It bears noting, however, that most BPEL compiler implementations inject or allow some proprietary and non-standard constructs, for example for Java snippets or accessing BPEL variables from Java, which reduces portability.

5.1.2 Advantages of BPEL

The real potential power of the above workflow technologies lies not only in the underlying BPEL foundation and its layering on widely-adopted web service standards, specifically WSDL which provides for type-neutral synchronous and asynchronous (callback-based) process interaction, but also in the higher-level capabilities built on top of it for business process modeling including task lists and human-in-the-loop interaction nodes, visual process design, and dashboard-based workflow monitoring and management. The ease and power of BPEL technologies for tying together web services to form new services and their implicit support for web service instance management and callbacks, as well as the possible interfacing with BPEL-based designers for composing workflows, make them potentially ideal candidates for application to orchestrating collections of information transformation services, both at the level of a runtime environment for the transformations as well as for building and organizing them.

In particular, BPEL is potentially well suited for fuselet development: its process model supports web service instance management and so allows getting handles to running fuselets, visual editors provide for flow control and XPath-based field selection, and runtime engines can support dashboard-based visual monitoring and control. Explorations to date lead to a belief that a BPEL engine might be extremely attractive in building a fuselet runtime environment: tailored user-interfaces can readily be built (e.g., using Java Server Pages) for process monitoring and management. Moreover, the tasklist-based workflow capabilities of BPEL, replete with task nodes for human interaction (e.g., approval using fill-in forms) are very advantageous, and would be ideal in modeling the IMS-staff notification of fuselet demands and the spawning and control of fuselets. The above capabilities, in particular the BPEL capabilities for human workflow interaction and detailed process monitoring and control (which will be largely untapped as they are beyond the scope of this effort), make a BPEL-based fuselet runtime environment worth consideration. It also might be feasible to integrate the semantics-based demand-driven fuselet inference capabilities being developing as part of this research effort into a Visual BPEL Designer Eclipse plug-in, as well as tie them into IMS notification so they can approve fuselets required to run under backwards chaining.

5.1.3 Candidate Technologies

Two principal implementations were considered as viable candidates to implement the BPEL demonstrations. The first was the ActiveBPEL engine, an open source BPEL engine from Active Endpoints, but whose associated visual process designer and runtime dashboards were at that time proprietary. Another candidate was the then newly released Oracle BPEL Process Manager for JBoss and the Oracle BPEL Process Designer plug-in for Eclipse. The licensing rights for Oracle's BPEL are currently limited to single prototype application, which is acceptable for the scope of this effort and attractive as it aids experimentation with a visual editor, as long as compatibility of the BPEL demonstration code with ActiveBPEL is maximized. However, it bears noting that Oracle is leading an Eclipse project for a BPEL Process Designer that would be based on Oracle's BPEL Designer – this was announced at the end of the Java One 2005 conference – so in the near future a comparable fully public-domain BPEL process designer will become available.

5.2 Expressing Individual Fuselets in BPEL

Expressing an individual fuselet in BPEL consists of three steps: (1) capturing the fuselet instance API in WSDL (with start, stop, pause, and resume methods), (2) coding the process flow of the fuselet in BPEL (that is, its flow logic and use of CAPI publish-subscribe services), and (3) coding the manipulation functions in Java extensions or individual web services. The process flow using BPEL constructs together with the business logic comprises the fuselet executable workflow.

There are two key problem areas in coding fuselets at the BPEL level: in the areas of callbacks from the CAPI publish-subscribe when using it as a web service, and in the area of how to code the manipulation functions. the use of a visual process designer as well the capabilities of various BPEL engines and monitors is also worth to exploring.

In particular, as is described below in more detail, in order to effectively use the CAPI as a web service including callbacks for subscribe operations, a prerequisite was to first wrap the Java CAPI as a BPEL web services process for use with fuselets. This task in turn required developing a partner-link enabled WSDL as well as a BPEL implementation for the publish-subscribe CAPI.

Coding of fuselets in BPEL under Oracle's BPEL Process Manager is briefly described below. This task involved two steps. The first step is to wrap the Java CAPI as a BPEL web services process for use with the fuselets: specifically a set of abbreviated publish-subscribe interfaces within a single WSDL to support callbacks, and implemented the process as a BPEL workflow with Java snippets to cut through to the CAPI invocations was designed. The second step was to design and implement fuselets using the Eclipse-based BPEL Visual Process Designer, using Java snippets to encode the business logic manipulation functions while keeping the CAPI invocations at the level of BPEL partner-link invocations.

5.2.1 Representing the JBI CAPI as a BPEL workflow

As mentioned above, there are two key problem areas in coding fuselets at the BPEL level. First, callbacks are an issue when using the CAPI as a web service. (A second problem area, discussed in the next section, is how to code the manipulation functions.) In the area of callbacks, the CAPI and its web service endpoints when exposed as stateless services via J2EE are insufficient. One alternative solution is to use polling which is in the JBI CAPI. A second solution is to provide a blocking subscribe method, either through an augmented API or as a BPEL extension (i.e., Java snippet). A third solution is to code a BPEL extension that essentially provides the callback capability.

A fourth and more attractive alternative is to slightly reengineer the CAPI implementation and expose it as a BPEL service, so as to take advantage of BPEL capabilities for bi-directional conversations, including callbacks. Callbacks are fully supported in WSDL through one-way invocations, and are very succinctly captured in BPEL using EventHandlers or iterative invoke-receive pairs. As an alternative to techniques mentioned above, the CAPI is wrapped as a BPEL process to create the needed WSDL endpoints. This also places the CAPI within the sphere of BPEL monitoring, e.g., for resource usage.

Thus, the CAPI itself is exposed as a BPEL process, i.e., as a web service constructed in BPEL, rather than accessed through Java snippets or exposed as a web service for example through J2EE or the Web Service Invocation Framework (WSIF). To wrap the Java CAPI as a BPEL web services process for use with fuselets, a set of abbreviated publish-subscribe interfaces is defined in one WSDL to support callbacks, and the process is implemented as a BPEL workflow with Java snippets to cut through to the CAPI invocations. In particular the CAPI implementation in BPEL was coded using Oracle's BPEL Visual Process Designer. The advantages to the approach of exposing the CAPI as a BPEL process are that it enables the use of callbacks within BPEL web services, that it enables BPEL monitoring of CAPI invocations, and that it insulates the CAPI implementation from use by other fuselets within BPEL.

In particular, by insulating the CAPI as a BPEL process, the mechanics of its implementation (including such details as starting the server) are insulated and in effect provide a plug-in module capability. This plug-in capability allows one to explore using alterative CAPI implementations: in this effort a lightweight in-house JBI CAPI implementation was used, while in the future one might plug in the JBI CAPI reference implementation.

5.2.2 Representing the fuselet interface and implementation in WSDL and BPEL

Expressing a fuselet as a workflow involves both capturing its interface using WSDL and its executable logic using BPEL. The logic for individual fuselets can be expressed using BPEL flow constructs together with Java snippets for manipulation function extensions; the former in particular can invoke publish-subscribe operations in the JBI CAPI when exposed as an atomic service. Correlation sets can be used to provide a stateful fuselet instance identifier as well as sequential use of a single CAPI connection. Issues to be addressed include instrumenting the

fuselet with monitoring, callbacks, fuselet parameterization, and optimal ways to invoke the CAPI when exposed as a BPEL process.

The fuselets use Java snippet extensions to perform the base manipulation functions. An alternative is to embody the manipulation functions as separate web services that would enable their use in the visual process designer with a modified Eclipse plug-in that provides a palette of manipulation functions and ontology-based selection. However, in the area of manipulation functions, Java snippets (i.e., BPEL extensions) are very easy to code and are integrated into the Eclipse BPEL designer. They are also significantly faster than making the manipulation function a web service, and are more readable, although the latter as well as using the Web Services Invocation Framework (WSIF) are still viable alternatives.

An advantage of expressing fuselet business logic in BPEL is the ability to exploit commercial, off-the-shelf BPEL visual process editors and monitors. Using BPEL is also in line with the fundamental approach of modeling fuselets as web services, critical to interoperability as well as leveraging evolving distributed process management and workflow technologies built on web services.

5.3 Expressing Fuselet Collections in BPEL

For capturing fuselet collections as BPEL workflows, specifically in the synthesized outputs from workflow composition, the basic technique is to express composition as concurrent invocations, i.e., a BPEL flow construct, and to use correlation sets to manage fuselet instances. Reducing fuselet collections to concurrent execution is not surprising given that fuselet interaction is essentially through a shared memory model, to wit, publish-subscribe. However, the above approach does not preclude the use of more advanced BPEL features to capture fuselet collections if so desired.

5.4 Demonstration of Fuselet Workflows in BPEL

To validate the above design of fuselets as BPEL processes, a concrete example of a generic fuselet in BPEL was developed that can be parameterized for tracking and threat prediction fuselets as well as a sensor publisher client. The BPEL and WSDL for the generic fuselet, as well as the BPEL and WSDL for the publish-subscribe CAPI implementation, were developed and run on the Oracle BPEL Process Manager platform.

In particular the example BPEL fuselet is a generic fuselet template that is parameterized to optionally subscribe to a topic (i.e., JBI information object type) with a selector predicate, to perform one of several transformations, and to optionally output the result to a topic subject to an outputCondition constraint. (The outputCondition was not implemented in the prototype.) The only domain-specific coding is in the Java snippet for the transformation that, based on the parameterization, executes one of several transforms, either through an inline switch statement or possibly using a set of transform services. The same template can thus operate as a sensor generator (no input), a transform (with input and output), or potentially a subscriber client (no output). The generic fuselet template thus allows one to chain together information transforms: in this case a chain is formed which pipes the sensor outputs to the tracking fuselet and from

there to the threat fuselet. The below screenshot illustrates how the generic fuselet can be instantiated using the parameterizations to perform a given transform. The sphere parameter is intended to represent a Community-of-Interest.

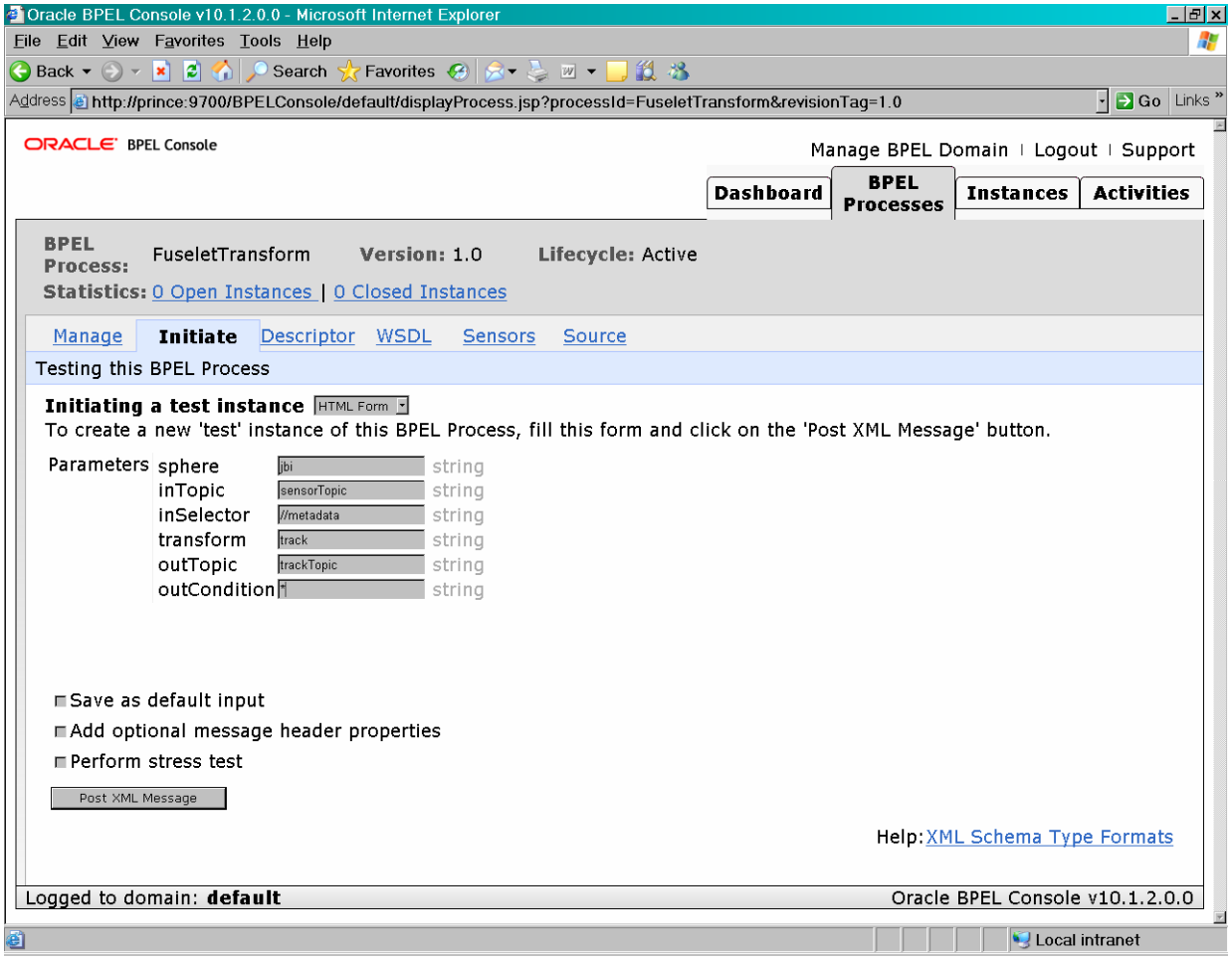

Figure 2. Instantiation of the Generic Fuselet Template

The fuselet WSDL is also of a generic form that includes methods to start, stop, pause, and resume the fuselet. The WSDL includes a BPEL partnerlink definition so that it is self-contained within BPEL. The template implements the above methods quite simply as EventHandlers that operate concurrently with the fuselet invocation. The power of BPEL as a concurrent programming language is illustrated in the simplicity of expressing the above controlling capabilities in a succinct and intuitive way. A similar template can be used by the workflow composition engine to synthesize and piece together fuselet invocations.

The below screenshot from the Oracle BPEL Process Designer illustrates the partnerlinks and methods exposed in the fuselet as well as the CAPI interfaces.

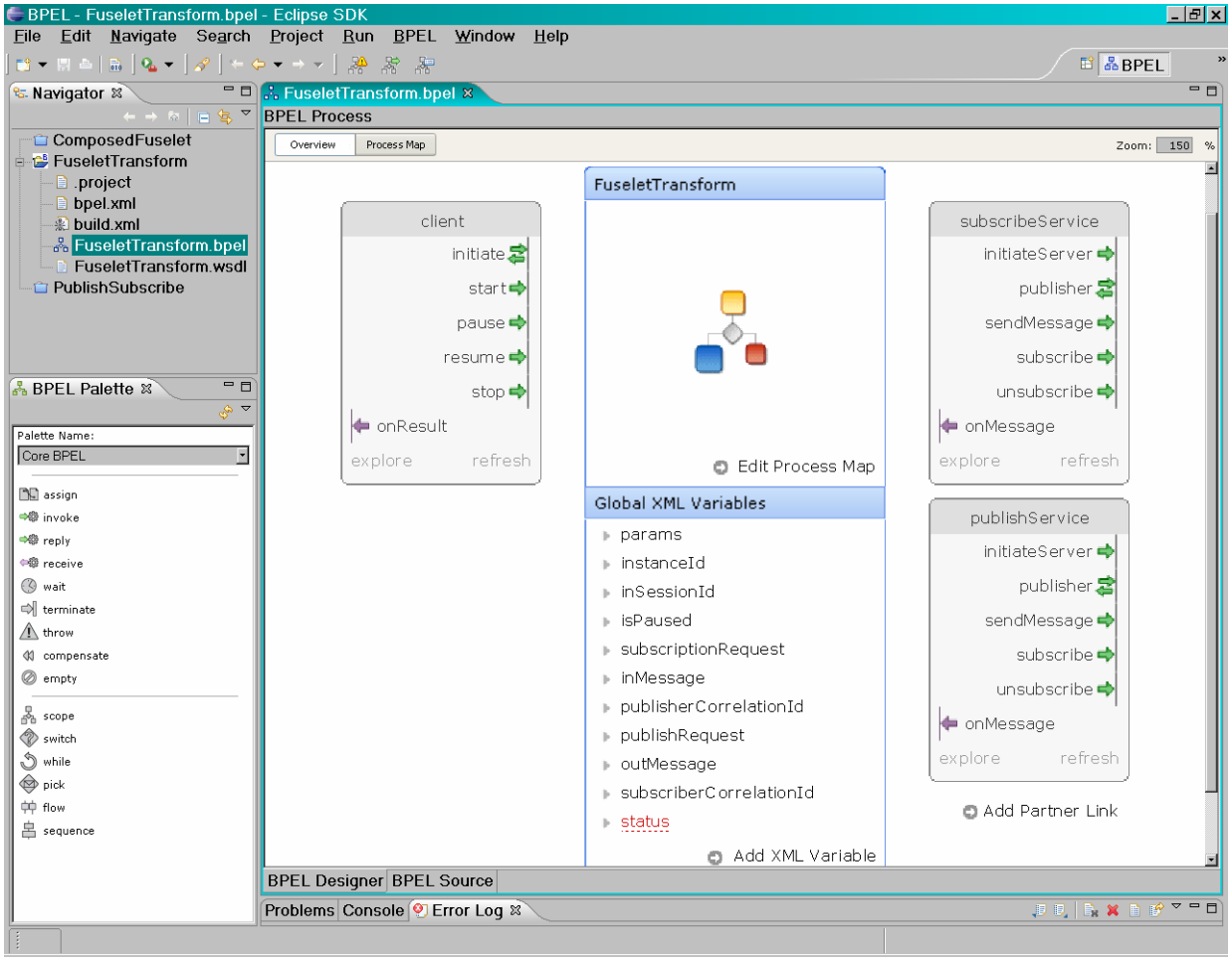

Figure 3. Fuselet Partnerlinks and Interfaces

Appendix A provides detailed examples of the CAPI and generic fuselet BPEL implementations.

6. Workflow Composition Engine

6.1 Overall Design

The objective of this portion of the research effort was to design and implement a workflow composition engine that takes a set of ontology-based fuselet specifications and a goal (also a fuselet specification), and synthesizes a BPEL workflow that invokes the fuselet BPEL implementations to achieve the goal. In previous sections a fuselet specification that uses Prolog-like Horn clause logic was described with ontology subtyping to express what a fuselet does; sample BPEL workflows were also developed that provided executable models for how the fuselet does it. In particular the latter evolved a generic BPEL template for fuselets that optionally subscribe to events, perform one of several transformations, and then optionally publish an output.

Given the above results, the design of the workflow composition engine is to (1) take the goal and the extracted Prolog-like portions of the fuselet specification, (2) run a backwards-chaining Prolog-like inference engine enhanced with ontology-subtyping for unification, and (3) extract the process model from proof, specifically extract a set of dependent parameterized fuselet invocations that are injected into a BPEL template for assembling fuselets.

6.2 Candidate Technologies

Key to implementing the above design is the choice of inference technology. In particular, the inference engine should support:

- Java integration
- Backwards-chaining (Prolog-like inference)
- Procedural annotations (side-effects, for example that could extract fuselet invocations from rule matching)
- Ontology subtyping

There are a number of inference technologies in the public domain. Jess (Java Expert System Shell) is one exception that is widely used but is not fully public domain, and does not explicitly support procedural annotations nor handle ontologies. Drools is a very nice object-oriented forward-chaining system that allows rules to be expressed in Java or Groovy, and is to be absorbed by the JBoss project; however, it does not support backwards-chaining. The Jena Semantic Web Toolkit has a pluggable inference engine, the default implementation of which is based on Declarative Logic (which includes notions of properties and subtyping), but its rules are also expressed in that logic. SweetRules (Semantic Web-Enabling Technology) is a translator from SWRL/RuleML onto various substrates including Jess and Jena. Mandarax is a Java library for building rules for backward-chaining including procedural annotations. Prova is a layer on top of Mandarax for a Prolog-like syntax for rules that also supports Java procedural annotations.

None of the above technologies directly support unification with ontology subtyping, that is, where a term publish(S(x)) matches publish(T(x)) if S is a subtype of T, although SWRL and SweetRules can indirectly express this using OWL predicates. Ontology subtyping is needed either directly or indirectly in workflow composition, for example, because if one fuselet publishes an information object with a type or selector that is more refined than that needed by the subscriptions of another fuselet, then in a demand-driven situation it is a viable candidate to be spawned off to supply the required information.

There are at least three ways to support unification with ontology subtyping. First, one can modify the unification algorithm: in Mandarax this is reasonably possible to do, and would also involve calling down into Jena. An alternative and fairly commonly used technique is to translate subtyping into explicit "is_a(S,T)" clauses in the rules. (This is also done in some of the literature on workflow composition using BPEL and ontology subtyping.) If the latter technique is chosen, the "is_a(S,T)" check involves invoking Jena to check if the subtyping relation holds in the ontology. Alternatively, one could go a step further and avoid using Jena entirely by translating the ontology subtyping relations into populated facts of above "is_a(S,T)" form; or, if using an object-oriented rule system (such as Drools, albeit in a forward fashion), by translating the ontology subtyping relations into object inheritance. (Both of these approaches are fairly straightforward and have appeared in the literature.)

There are two viable implementation alternatives. The first is to use SweetRules with its Jena translator: this is attractive because it uses RuleML (which is also the foundation of the Semantic Web Rule Language (SWRL) draft W3C standard), but it is a bit difficult to express the necessary procedural annotations. The second alternative is to use Mandarax/Prova and either translate the ontology database into explicit subtyping facts, or better yet to translate the fuselet specification rules into a form that use an "is_a(S,T)" relation that in turn cuts through to Jena for determining subtyping relations from the ontology. The latter is both feasible and attractive because Prova is highly integrated with Java, and the technique is easy to implement and understand. In addition, if a technology like Janino is used for dynamically compiling Java snippets (Janino is used by Drools), or alternatively a Java-based scripting language such as Beanshell is employed, a prototype can be built that is totally scripted, easy to understand, and requires no compilation. The latter was deemed a good approach for this research effort.

On the basis of the above analysis, Mandarax/Prova was selected for the Java-based backwards-chaining inference technology. The implementation of the workflow composition engine thus consists of (1) XSLT scripts to extract the fuselet specifications and introduce the "is_a" relation in the rules for ontology subtyping, (2) the use of Mandarax/Prova for the inference engine, and finally (3) a step to integrate the derived fuselet invocations into a BPEL template.

6.3 Supporting Ontology-Based Subtyping in Inference

In order to incorporate ontology-based subtyping in inference, the original fuselet specifications of the form:

```
publish (topic, selector, transform(X,Y)) :-
```

```
            subscribe (topic1, selector1, X), subscribe (topic2, selector2,
Y).
```

are translated, thus allowing unification across literal terms which are ontology subtypes, to a form which incorporates an explicit "is_a" relation to capture ontology subtyping relations:

```
      publish (topic, S, transform(X,Y)) :- is_a(selector, S),
            subscribe (topic1, selector1, X), subscribe (topic2, selector2,
Y).
```

which allows conventional unification, where only the uppercase variables will be unified.

In the first part of the above example the intended meaning is that, for the right-hand side term “subscribe(…selector1…)”, if there is another rule whose head has a selector term that is a refinement of "selector1" (where the selectors are predicates in publish-subscribe), then during backwards chaining "publish(...selector1...)" should unify with that term[1]. The following is an informal explanation of what such unification means. In the above rules (for which there may be several per fuselet), the right-hand side represents subscriptions consumed by the fuselet, and the left-hand side represents publications produced by the fuselet. In backward chaining, or demand-driven search, the data items needed by a fuselet can be seen on the right-hand side, and they are searching for another fuselet that would produce the required entries. If another fuselet produces data that is more refined that what is needed (i.e., is a subtype), then that is sufficient for the input; however, if the produced type is a mismatch or less refined then it would not meet the subscription predicate.

Thus the backwards-chaining inference, or Prolog-style proof, from a goal of the form:

```
      :- subscribe(topic3, selector3, X)
```

represents a search for a chain of fuselets that would produce the required information: the workflow to realize the original goal then includes instantiation of all fuselets used in the backwards-chaining proof. It also bears noting that typically there would need to be "facts" in the rule database that let the proof successfully conclude its search; such facts are assertions of the form:

```
      publish(topic4, selector4, data)
```

that represent published data already assumed to exist, and thus for which no fuselets need to be spawned to create it, or more generally a statement of what fuselets are already running.

Typically one might expect the predicates, or message selectors, to be expressed in a language such as XPath. However, there is good reason for not directly using such a notation in the logical relations captured above: comparing XPath expressions in order to determine subtyping is somewhat complicated and often will yield trivially exclusive results. The above approach separates the problem of deciding which fuselet inputs match other fuselet outputs into two stages: a first stage that maps XPath-based input selectors into ontology terms

[1] The publish and subscribe literals are allowed to unify though a ancillary rule that states: if publish() then subscribe().

(further described in Section 2), and a second stage that uses those terms at a well-defined logical level to reason about fuselet composition and needs satisfaction.

The implementation thus uses an OWL ontology of selector subtypes in reasoning about predicate refinements. Prova, which is a nicely bundled package that runs out-of-the-box when unzipped, includes Jade and almost all of the Jena Semantic Web Toolkit. A small Jena wrapper class was written that provides an "is_a" relation based on ontology subtyping, and this used in logic statements of the above form. The above form is a fairly standard way to encode ontology-based subtyping in Prolog-like reasoning without altering the underlying unification algorithm to use subtyping in term matching. Alternatively, instead of using ontology-based subtyping via Jena one could use Prova's support for respecting Java class inheritance when performing unification of variables which are Java objects, and encode the subtyping relations in a Java class hierarchy.

6.4 Detailed Architecture

The architecture of the workflow composition engine, in greater detail, is illustrated below in Figure 2.

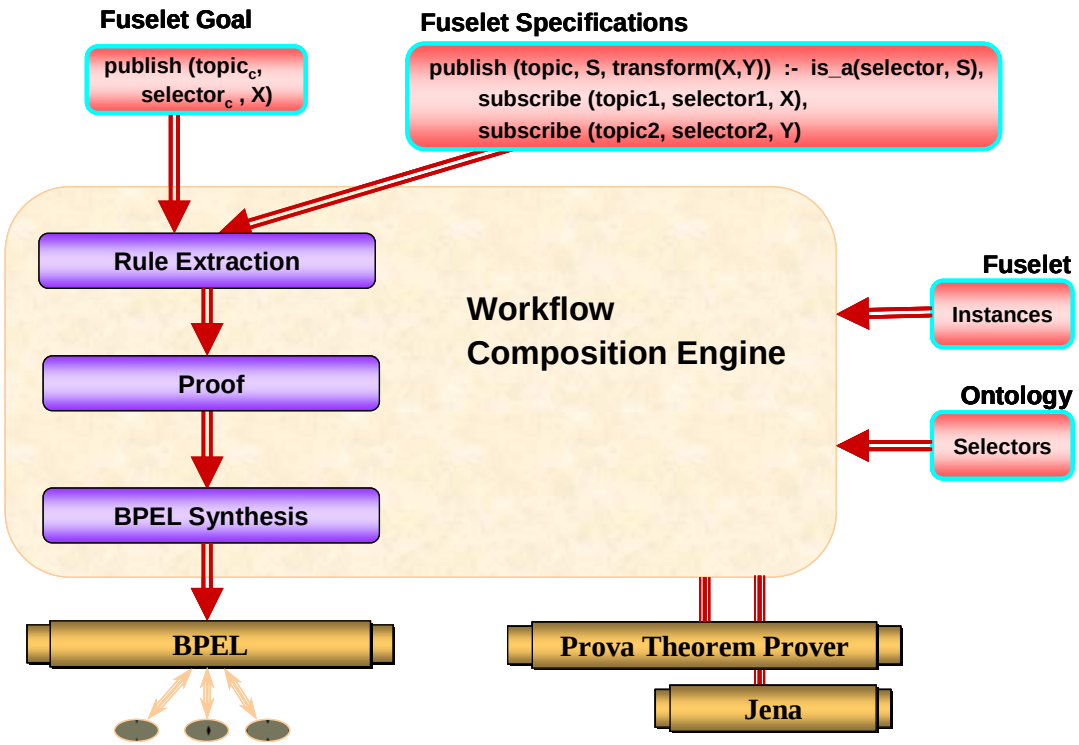

Figure 4. Architecture of the Workflow Composition Engine

The workflow composition engine consists of three phases: (1) extracting the specifications and goal into a Prolog-like form with explicit ontology-based subtyping, (2) running Mandarx/Prova to infer the required fuselet invocations required to satisfy the goal, and (3)

synthesizing the BPEL invocations, in particular embedding them in a generic BPEL template for assembling fuselets. In Figure 4, for purposes of illustration the specification in the upper right is shown with the "is_a" relation already introduced; however in actuality this is introduced in the rule extraction stage.

The implementation is coordinated by a Java controller class which performs the above stages. The software implementation, in addition to the Jena wrapper mentioned previously, provides a small Java wrapper class to call Prova using its Java interface. The implementation invokes Prova through this wrapper to perform the reasoning involved in workflow composition.

6.5 Metrics and Feedback Mechanisms for Assessment and Reengineering

The following discussion briefly examines how metrics and feedback mechanisms could be expressed and implemented respectively for guiding the re-engineering of fuselet workflow models to enable continuous fuselet improvement and maximize the value of information produced by fuselets.

Within the ontology-based framework, metrics would map XPath-selectors characterizing published results to their equivalent ontology terms in the specification. Such metrics might be captured within the OWL-S fuselet specification as a Quality-of-Service (QOS) section of the service profile, in the form of a map from sets of XPath selectors to ontology terms for a given fuselet. For example, a metric that maps track selectors might appear as:

```
<Metric>
  <MapEntry>
    <key>//metadata[type='track'][sensorRadius<'3']</key>
    <value>RefinedTrackSelector</value>
  </MapEntry>
  <MapEntry>
    <key>//metadata[type='track']</key>
    <value>TrackSelector</value>
  </MapEntry>
</Metric>
```

Figure 5. Metric Mapping XPath Selectors to Ontology Terms

Such metrics can then be used to incorporate feedback mechanisms into re-engineering the workflow orchestration to optimize needs satisfaction, by: (1) injecting sensors into the BPEL execution substrate to monitor the fuselet outputs, perhaps through BPEL aspect-oriented instrumentation, (2) running an assessment agent or analyzer which takes the monitored outputs, maps them into ontology terms using the metrics in the service profile, and then compares the ontology terms with the expected selectors in the fuselet production rules, (3) running a workflow composition agent or planner that, upon event notifications from the assessment agent, re-engineers a fuselet workflow to provide an alternative implementation, and (4) running an execution agent to implement the changed plans.

Figure 3 illustrates how workflow composition fits into the above larger feedback mechanism loop. In addition there might be an additional controller agent to manage the lifecycle and connections of the above sensors, analyzer, planner, and executor.

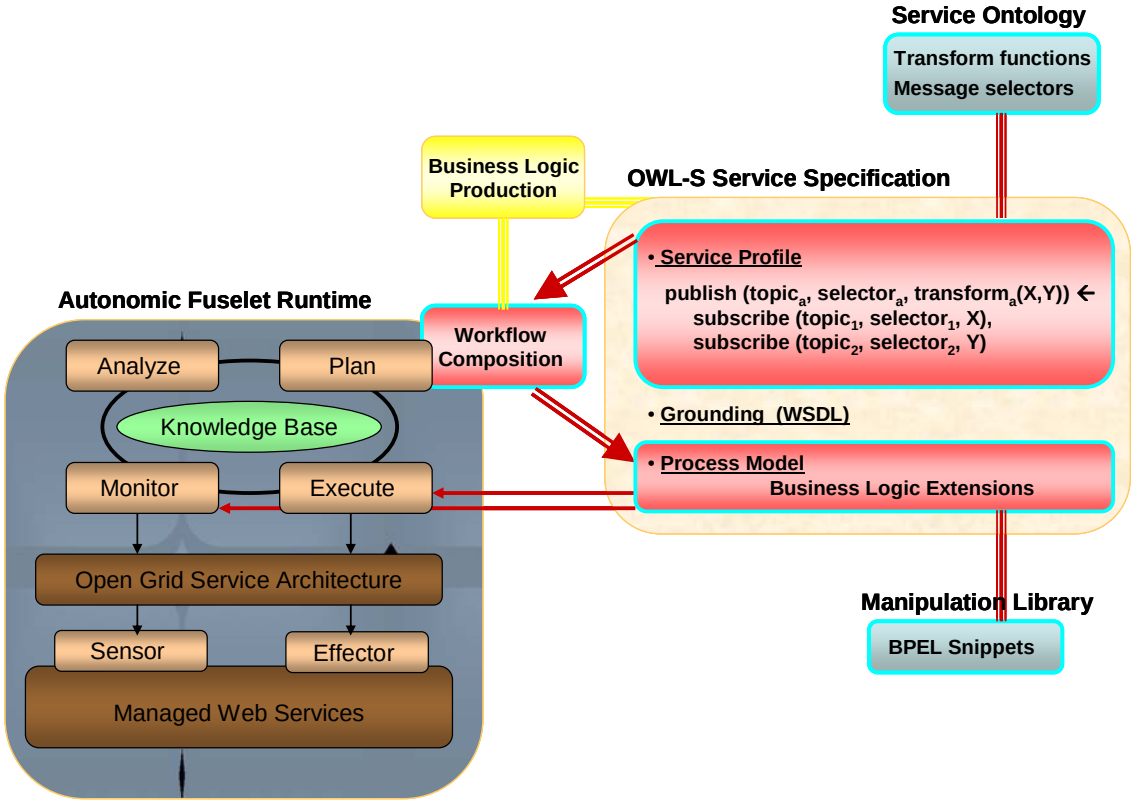

Figure 6. Autonomic Fuselet Assessment and Re-Engineering

The re-engineering, or planning step described above basically entails either choosing among alternatives in the proof path or altering the underlying facts or rules. Re-planning can be accomplished in several ways: by remembering the exact proof or derivation of the fuselet associated with a given output performance and then searching for an alternative proof; by altering the metrics themselves or the fuselet predicates; or by re-examining the composition problem with a different set of admissible fuselets which are perhaps prioritized according to some expected QOS. The above alternative search mechanisms minimally require slightly modifying the composition engine so that it avoids or prefers certain search paths: this could be accomplished for example by either re-synthesizing the rules to reflect past results using cut operators, by tailoring the underlying Mandarax engine, or preferably by running Prova in exhaustive mode (rather than "solve" mode) which yields all possible proof derivations, and managing them explicitly. Sophisticated techniques for re-planning might also be incorporated using HTN planners, as is mentioned in the conclusions.

6.6 Example of Workflow Composition

An example of workflow composition is provided in Appendix A.

Tracking and threat fuselets, as well as a sensor subscriber client, have their needs specified using OWL-S and their executable business logic specified using BPEL.

Using workflow composition, a third fuselet is synthesized from a goal specified in OWL-S to subscribe to a refined threat alert. The tracking and threat fuselets are initiated within the workflow, but not a sensor publisher which is recognized from a set of instance facts as already running. The composed fuselet thus acts as a workflow container for other fuselets; however, its WSDL exposes the same interfaces as a fuselet, and it could additionally subscribe, perform a transformation, and conditionally publish an output. It bears noting that the workflow composition engine in its prototype form does not generate the OWL-S specification for the composed fuselet workflow: however, this is a relatively straightforward extension.

7. Conclusions

In this research effort a proof-of-concept demonstration of autonomic fuselet business logic development was developed that uses semantic web services and workflow technologies to specify fuselet information needs, to define an executable workflow model for its business logic, and to perform workflow composition from a goal and specifications to executable model in a manner that enables feedback mechanisms that can assess behavior and then reengineer the process model to satisfy needs.

First, an ontology-based fuselet specification in a variant of the Web Ontology Language for Services (OWL-S) was defined. Then the technical feasibility of capturing fuselet business logic using the Business Process Execution Language (BPEL) was examined, in particular the suitability of BPEL for controlling fuselets as well as interacting with the publish-subscribe API (CAPI). Lastly, a workflow composition engine for deriving a BPEL process model from a goal and a set of fuselet specifications was designed and implemented. Also examined were how metrics and feedback mechanisms could be used to enable assessment and re-engineering.

A fundamental observation that underlies this approach is that it may be possible to increase the robustness of general techniques in semantic web services and workflow composition by restricting the domain of discourse to the fundamental characteristics of fuselets, i.e., that they interact through publish, subscribe, and query operations. An otherwise general OWL-S specification is thus restricted to a simple Horn-clause logic formulation whose terms are publish-subscribe operations, and this formulation is leveraged in workflow composition by using Prolog-based theorem proving enhanced with ontology subtyping.

7.1 Lessons Learned

For specifying fuselet information needs, ontology-based Prolog-like logic statements are a succinct means of capturing fuselet inputs and outputs as well as their logical relation in a manner that is amenable to inference. Additionally, in can be concluded that OWL-S is well-suited for capturing ontology-based fuselet specification. Evolving RuleML and SWRL standards are highly expressive, and the service profile and process model provide a clean separation of concerns. Moreover, as is discussed below, the use of OWL-S enables the expression of more sophisticated fuselet behaviors, as well as the integration of those specifications into sophisticated Hierarchical Task Network planners.

In the area of fuselet business logic, BPEL is one viable medium for capturing fuselet executables, both at the level of individual fuselets as well as for fuselet collections. BPEL provides a rich asynchronous programming model, which when coupled with a powerful visual process designer (such as the BPEL visual designer Eclipse project) has the potential to provide a fuselet Interactive Development Environment. One drawback of BPEL, in comparison for example to the XML Process Definition Language (XPDL), is in its lack of support for directly invoking other BPEL processes as subflows rather than as encapsulated web services. However, this shortcoming is addressed in proposals by IBM and others for sub-process extensions in BPEL-SPE.

For workflow composition, a simple specification based on Horn clause logic has the advantage of being amenable to reasoning using conventional theorem provers and rule engines. In particular, Prova/Mandarax is an excellent and easy-to-use backwards-chaining rule inference system that provides Java integration and bundled support for the Jena Semantic Web toolkit. However, more sophisticated specifications or planning techniques entail the use of concomitantly more sophisticated semantic rule engines and AI planners.

7.2 Recommendations for Future Work

A key result of this research effort is the identification of techniques and technologies that can be used for semantics-based fuselet specification and workflow composition

While a simple fuselet specification using Horn-clause logic with ontology subtyping appears to be a good match for capturing fuselet behavior, and increases the robustness of planning whatever the technology, one should not preclude the expression of more complex behaviors using a fuller set of the SWRL rule language as well as the use of more features of OWL such as properties and restrictions. The expression of more sophisticated behavior in SWRL in turn may require declarative logic (DL) reasoners such as the SweetRules translator or Pellet.

In addition, while the planning used in the workflow composition implementation is straightforward, more sophisticated planning techniques may require the use of Hierarchical Task Planners (HTN) such as JSHOP2. In particular JSHOP2 includes an OWL-S API for interfacing with and performing planning using OWL-S specifications, which is made practical by the fact that OWL-S is at heart based on HTN planning concepts.

As a next step it is recommended that:

(a) OWL-S be used for fuselet specification, which supports the above Horn clause formulation (which is initially recommended and potentially increases the robustness of planning whatever the technology) as well as more complex specifications if needed,

(b) the JSHOP2 HTN planner be used for implementing workflow composition, in order to leverage and track COTS planning technology. This also supports more general OWL-S specifications as needed.

In the larger arena, the above semantics-based demand-driven fuselet inference capabilities should be integrated into fuselet development environments as well as into the fuselet runtime to provide advanced semantics-based fuselet authoring and autonomic runtime management capabilities. Specifically, the capabilities for ontology-based fuselet specification and composition can be integrated into a Visual BPEL Designer Eclipse plug-in, as well as be tied into IMS notification so they can approve fuselets required to run under backwards chaining.

To enable the full cycle of autonomic fuselet execution, one must be able to perform both metric-based fuselet assessment as well as perform web service composition in response to assessment. For the former, the fuselet specification metrics that map XPath-based selectors to a hierarchy of ontology terms need to be derived. From a practical perspective, the semi-automatic derivation of ontology subtyping of terms from their XPath selectors might be examined. For the latter, the

enhancement of the runtime to incorporate both assessment as well as planning is key to realizing the larger goal of autonomic execution.

References

Joint Battlespace Infosphere (JBI)

[CAPI[JBI Publish-Subscribe Common API (CAPI). http://www.infospherics.org

[Fuselets] Fuselet Definition Documents. http://www.fuselet.org

[JBI] AFRL JBI Program. http://www.if.afrl.af.mil/tech/programs/jbi/

[USAF99] "United State Air Force Scientific Advisory Board Report on Building the Joint Battlespace Infosphere, Volume 1: Summary", SAB-TR-99-02 December 17, 1999. http://www.sab.hq.af.mil/Archives/1999/JBI/JBIVolume1.pdf

OWL-S Documentation

[OWL-S] OWL-S: Semantic Markup for Web Services, W3C Member Submission, 22 November 2004. http://www.w3.org/Submission/OWL-S

[SWRL] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, 21 May 2004. http://www.w3.org/Submission/SWRL/

BPEL Documentation

[BPELSource] "BPEL Resource Guide". http://www.bpelsource.com/

[BPELSpec] "OASIS Web Services Business Process Execution Language (WSBPEL)". http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel

[BPEL11] "Business Process Execution Language for Web Services Version 1.1". http://www.ibm.com/developerworks/library/specification/ws-bpel/

Visual BPEL Designers

[BpelEclipse] Eclipse BPEL Designer Editor. http://www.eclipse.org/bpel

[BpelTutorial] Oracle BPEL Quick Start Tutorial - Eclipse. http://www.oracle.com/technology/products/ias/bpel/pdf/orabpel-QuickStart.pdf

BPEL Implementations

[ActiveBPEL] ActiveBPEL Engine - Open Source BPEL Server. http://www.activebpel.org/

[Oracle] Oracle BPEL Process Manager. http://www.oracle.com/appserver/bpel_home.html

Prova and Jena

[Jena] Jena Semantic Web Framework. http://www.hpl.hp.com/semweb/jena.htm

[Prova] Prova Language for Rule Based Java Scripting. http://www.prova.ws

Web Services Composition

[ACC04] "An algorithm for Web service discovery through their composition", Lerina Aversano, Gerardo Canfora, and Anna Ciampi, *Proc International Conference on Web Services 2004 (ICWS 2004).*
http://conferences.computer.org/icws/2004/
http://csdl.computer.org/comp/proceedings/icws/2004/2167/00/2167toc.htm

[CSD+03] "Business-oriented management of Web services", Fabio Casati, Eric Shan, Umeshwar Dayal, and Ming-Chien Shan, *Communications of the ACM*, Volume 46 , Issue 10 (October 2003). http://maximus.uvt.nl/sigsoc/pub/Papazoglou%20-%20Georgakopoulos%20-%20Service-oriented%20computing.pdf

[MWG04] "Automated Web Service Composition using Semantic Web Technologies", Shalil Majithia, David W.Walker, and W.A.Gray, *Proc. International Conference on Autonomic Computing (ICAC'04).*
http://csdl.computer.org/comp/proceedings/icac/2004/2114/00/2114toc.htm

[OWL-S] "Bringing Semantics to Web Services: The OWL-S Approach", David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara, *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, July 6-9, 2004, San Diego, California, USA.
www.daml.org/services/owl-s/OWL-S-SWSWPC2004-CameraReady.doc

[RKM04] "Logic-based Web Services Composition: from Service Description to Process Model", Jinghai Rao, Peep Kungas, and Mihhail Matskin, *Proc. International Conference on Web Services 2004 (ICWS 2004).*
http://conferences.computer.org/icws/2004/

Symbols, Abbreviations, and Acronyms

BPEL.....Business Process Execution Language
CAPI .....Common API for Publish-Subscribe (JBI)
COTS .....Commercial Off-The-Shelf
DL .....Declarative Logic
JBI.....Joint Battlespace Infosphere
KIF .....Knowledge Interchange Format
OWL .....Ontology Web Language
OWL-S.....Ontology Web Language for Services
QOS.....Quality of Service
RDF.....Resource Description Format
RDF-S .....Resource Description Format Schema
RuleML.....Rule Markup Language
SOAP .....Simple Object Access Protocol
SWRL .....Semantic Web Rule Language
UDDI.....Universal Description, Discovery, and Integration
WSDL .....Web Service Description Language
XML.....Extensible Markup Language
XPDL .....XML Process Definition Language

Appendix A. Demonstration Script.

A.1 Overview

A fuselet may be defined as a form of information transformation that communicates with the world using only publish-subscribe operations, for example as encapsulated in the Joint Battlespace Infosphere (JBI) Common Application Programming Interface (CAPI) for publish-subscribe. This document presents step-by-step instructions on how to display and run examples that demonstrate:

(1) The specification of fuselet information needs using the Web Ontology Language for Services (OWL-S).

(2) The definition of fuselet business logic using the Business Process Execution Language (BPEL), and its execution as a web service.

(3) The automatic composition of fuselets from other fuselets using a workflow composition engine that derives a BPEL process model from a fuselet goal and other fuselet specifications using theorem-proving.

In the following sections the demonstration scenario and objectives as well as the capabilities to be demonstrated are described. Lastly the demonstration programs themselves are listed.

A.2 Objective

The objective of these demonstrations is to show capabilities for autonomic fuselet business logic development through the use of semantic web services and workflow technologies to:

- Specify fuselet information needs,
- Define an executable workflow model for its business logic, and
- Perform workflow composition from a goal and specifications to executable model in a manner that enables feedback mechanisms that can assess and adapt the process model to satisfy needs.

Details on each of these objectives are provided below.

A.2.1 Ontology-Based Fuselet Specification

The objective of the first part of the demonstration is to show how the information needs of a fuselet can be defined through an ontology-based fuselet logic specification expressed in a variant of the Web Ontology Language for Semantic Web Services (OWL-S). Concrete fuselet examples are provided which use the OWL-S service profile to capture a Horn Clause (Prolog-like) logic specification with basic manipulation terms and message-selector types taken from an ontology. For each fuselet, its specification consists of a set of if-then statements of the form:

```
publish (topic, selector, transform(X,Y)) IF
   subscribe (topic1, selector1, X) AND subscribe (topic2, selector2, Y).
```

where

`topic:` is the JBI info-object type

`selector:` is the JBI selector predicate, where term matching uses ontology subtyping

`transform:` transformation, where term matching may optionally use ontology subtyping

`X,Y: are` logical variables

In order to incorporate ontology-based subtyping in inference, these clauses are represented in OWL-S using an explicit "is_a" relation to capture ontology subtyping relations:

```
publish (topic, S, transform(X,Y)) IF is_a(selector, S) AND
   subscribe (topic1, selector1, X) AND subscribe (topic2, selector2, Y).
```

which allows unification across literal terms which are ontology subtypes.

The intended meaning of ontology-based selectors is that the demand for subscription can be satisfied by another fuselet publishing a refined type. In the above rules (for which there may be several per fuselet), the right-hand side represents subscriptions consumed by the fuselet, and the left-hand side represents publications produced by the fuselet. In backward chaining, or demand-driven search, the right-hand side shows the data items needed by a fuselet, and are searching for another fuselet that would produce the required entries. If another fuselet produces data that is more refined that what is needed (i.e., is a subtype), then that is sufficient input; however, if the produced type is a mismatch or less refined then it might not meet the subscription predicate.

The above logical formulae are encoded directly in the Semantic Web Rule Language (SWRL) used by OWL-S for specifying Inputs/ Outputs/ Preconditions/ Effects. Such a notation thus enhances fuselet metadata that might otherwise only specify inputs and outputs by additionally providing preconditions and effects. In particular, the above representation of fuselets only uses a small subset of the OWL-S and SWRL capabilities for expressing rules: in the general case the conditions can be more sophisticated declarative logic rules to capture a finer characterization of fuselet capabilities.

Such a specification can then be used both to:

- Assess whether the inputs and outputs of the executing fuselet business logic satisfy the specified needs, and
- Perform automated fuselet composition using theorem provers or planners, with inference mechanisms enhanced to use ontology-based subtyping.

The above notation has the advantage of separating the decision as to which fuselet inputs match other fuselet outputs into two stages: a first stage that maps XPath-based input selectors into ontology terms (which can be recorded in the ontology or in the fuselet specification as a

nonfunctional attribute), and a second stage that performs ontology-aware reasoning. The separate stage that maps from predicates, e.g., XPath expressions, to selector terms could be captured in the OWL-S service profile as quality-of-service metric.

In addition to using the OWL-S service profile to capture the information needs, the two other pieces of OWL-S -- the process model to hold the business logic, and the grounding to hold the mapping to WSDL -- are used. To enable workflow composition, the grounding is included as part of the fuselet specification.

The demonstration provides:

(1) An example ontology of selector predicates (which could also hold manipulation functions), and

(2) Example fuselet specifications in OWL-S for tracking and threat prediction fuselets, as well as a sensor publisher client.

A.2.2 BPEL-Based Fuselet Business Logic

The second part of the demonstration shows how to map individual fuselets, as well as their composition, into BPEL. A concrete example of a fuselet running on a COTS BPEL execution platform is demonstrated.

Expressing a fuselet as a workflow involves both capturing its interface using WSDL and its executable logic using BPEL. The logic for individual fuselets can be expressed using BPEL flow constructs together with Java snippets for manipulation function extensions, where the former can invoke publish-subscribe operations in the JBI CAPI when exposed as an atomic service. Correlation sets can be used to provide a stateful fuselet instance identifier as well as sequential use of a single CAPI connection. Issues that need to be addressed include instrumenting the fuselet with monitoring, callbacks, fuselet parameterization, and optimal ways to invoke the CAPI as a service endpoint.

In particular the CAPI itself is exposed as a BPEL process, i.e., as a web service constructed in BPEL, rather than accessed through Java snippets or exposed as a web service for example through the Web Service Invocation Framework (WSIF). The advantages to exposing the CAPI in BPEL are that it allows the CAPI implementation to take advantage of BPEL capabilities for bi-directional conversations, including enabling the use of callbacks within BPEL web services, and that it enables BPEL monitoring of CAPI invocations (e.g., for resource usage). In particular, callbacks are fully supported in WSDL through one-way invocations, and are very succinctly captured in BPEL using EventHandlers or iterative invoke-receive pairs. To wrap the Java CAPI in a BPEL web services process, an abbreviated set of publish-subscribe interfaces is defined in one WSDL to support callbacks, and the process is implemented as a BPEL workflow with Java snippets to cut through to the CAPI invocations.

An advantage of expressing fuselet business logic in BPEL is the ability to exploit COTS BPEL visual process editors and monitors. Using BPEL is also in line with the fundamental

approach of modeling fuselets as web services, critical to interoperability as well as leveraging evolving distributed process management and workflow technologies built on web services.

The demonstration provides:

(1) A BPEL implementation of the CAPI, specifically WSDL and BPEL code for an abbreviated set of CAPI publish-subscribe interfaces,

(2) A generic fuselet business logic definition in WSDL and BPEL that can be parameterized for tracking and threat prediction fuselets as well as a sensor publisher client,

(3) Instructions to start the COTS BPEL server,

(4) Instructions to compile the example fuselet, and

(5) Instructions to execute a chain of instantiated fuselets.

In particular the example BPEL fuselet is a generic fuselet template that is parameterized to optionally subscribe to a topic (i.e., JBI information object type) with a selector predicate, to perform one of several transformations, and to optionally output the result to a topic if it satisfies an output condition. The only domain-specific coding is in the Java snippet for the transformation that, based on the parameterization, has a switch statement to execute one of several transforms. (This inline coding can be generalized to instead use a transform factory, or a transform web service, that allows arbitrary transforms.) The same template can thus operate as a publisher generator (no input), a transform (with input and output), or potentially a subscriber client (no output).

The fuselet WSDL is also of a generic form that includes methods to start, stop, pause, and resume the fuselet. The WSDL includes a BPEL partnerlink definition so that it is self-contained within BPEL. The template implements the above methods quite simply as EventHandlers that operate concurrently with the fuselet invocation: the power of BPEL as a concurrent programming language is illustrated in the simplicity of expressing the above controlling capabilities in a succinct and intuitive way.

A similar template is used by the below composition engine to piece together synthesized fuselet invocations.

A.2.3 Workflow Composition Engine

The third part of the demonstration shows the capability to automatically perform workflow composition. The workflow composition engine composes fuselets from other fuselets by deriving a BPEL process model from a fuselet goal and a set of fuselet specifications expressed in OWL-S. In particular the demonstration shows the use of ontology-based subtyping in inferring how the demand for a subscription can be satisfied by another fuselet publishing a refined type.

The workflow composition engine is a Java package that leverages COTS technologies such as the Prova/Mandarax inference engine and the Jena Semantic Web Toolkit to perform ontology-aware theorem proving from the goal which as a side-effect extracts the process model from the proof. Its inputs, outputs, and algorithm as are follows.

The inputs are:

- A fuselet goal in the form of an OWL-S specification.
- Existing fuselets specified in OWL-S.
- An OWL ontology containing a hierarchy of selector terms.
- A database of existing running fuselet instances, as Prova rules.

The composition engine uses ancillary files including:

- Supporting Prova rules, e.g., append.
- A BPEL composition template.

The composition engine outputs a BPEL workflow that invokes a fuselet web service for each derived fuselet invocation, using the previously described parameterized generic fuselet BPEL template. The synthesized workflow may then be compiled with Oracle BPEL (using "obant") and subsequently run.

The algorithm to perform workflow composition from goal and specification to invocation set roughly consists of three steps: extract rules, perform proof, and synthesize the BPEL template. In more detail the algorithm is as follows:

- Extract the goals and specifications into Prova rules using XSLT.
- Form a combined rule set using the extracted goals, specifications, instance database, and supporting rules.
- Run Prova on the rules to infer fuselet invocations.
- Generate BPEL fuselet invocations from the inference results.
- Insert the BPEL invocations into the template.

The package is structured as a set of independent Java source files for XSLT Transformation, Jena invocation, and Prova invocation that may be used separately from workflow composition.

The above technique for workflow composition is one prerequisite component for an autonomic fuselet runtime, in that it is intended to be used in concert with feedback mechanisms that compare the executed fuselet business logic outputs with metrics for the specified needs, and then dynamically re-engineer the derived fuselet model using workflow composition as needed to optimally satisfy the needs.

The demonstration provides:

(1) An example fuselet goal specified in OWL-S,

(2) A database of existing running fuselet instances, expressed as Prova rules,

(3) A script to execute the workflow composition engine to create the composed fuselet process model from: the goal, the set of fuselet specifications in OWL-S for tracking and

threat prediction fuselets and sensor publisher client, and the instance database, given the ontology of selector predicates,

(4) The resulting synthesized fuselet business logic,

(5) Instructions to compile the composed fuselet,

(6) Instructions on how to execute the composed fuselet as well as the sensor generator client.

A.3 Scenario

The scenario of the demonstration is in the area of target tracking and threat evaluation, specifically to create and run fuselets that use the JBI CAPI to publish and subscribe to sensor, target tracking, and threat prediction events.

In the demonstration scenario, one JBI client publishes sensor returns, a Tracker fuselet subscribes to and uses the sensor returns to update track locations, and another Threat fuselet subscribes to and evaluates the updated tracks to publish threat alerts.

The two tracking and threat fuselets, as well as the sensor subscriber client, have their needs specified using OWL-S and their executable business logic specified using BPEL.

Using workflow composition, a third fuselet is synthesized from a goal specified in OWL-S to subscribe to a refined threat alert. The tracking and threat fuselets are initiated within the workflow, but not a sensor publisher which is recognized from a set of instance facts as already running. The composed fuselet thus acts as a workflow container for other fuselets; however, its WSDL exposes the same interfaces as a fuselet, and it could additionally subscribe, perform a transformation, and conditionally publish an output.

In particular the demonstration shows the use of ontology-based subtyping in inferring how the demand for a subscription can be satisfied by another fuselet publishing a refined type. The ontology subtyping occurs both in matching the goal subscription selector ("threatSelector") to the threat fuselet publication condition ("refinedThreatSelector"), and in matching the threat fuselet's track subscription selector ("trackSelector") to the track fuselet publication condition ("refinedTrackSelector"). The ontology subtyping is illustrated below in the fuselet specifications, goal, and instances, the former which are extracted from the OWL-S in readable form by the composition engine.

```
%-------------------------------------------------------
% Extracted rules from specifications in readable form
%-------------------------------------------------------

% Threat fuselet
%---------------
publish( threatTopic, "refinedThreatSelector", threat( X)) :-
   subscribe( trackTopic, "trackSelector", X).

% Tracker fuselet
```

```
%---------------
publish( trackTopic, "refinedTrackSelector", track( X)) :-
   subscribe( sensorTopic, "sensorSelector", X).
publish( trackTopic, "trackSelector", track2( X, Y)) :-
   subscribe( sensorTopic, "sensorSelector", X),
   subscribe( sensorTopic2, "sensorSelector2", Y).

% Sensor fuselet
%--------------
publish( sensorTopic, "sensorSelector", sensor( data)).

%-----
% Goal
%-----
:- subscribe( threatTopic, "threatSelector", X).

%-----------------------------
% Facts about fuselet instances
%-----------------------------
exists_fuselet(sensorFuselet, [exists, sensorFuselet,
   ["jbi", "none", "none", "sensor", "sensorTopic", "*"]).
```

Figure 7. Extracted Fuselet Specifications, Goal, and Instances in Readable Form.

The demonstration shows:

(1) Execution of the individual fuselets on a COTS BPEL engine,

(2) Workflow composition, and

(3) Execution of the composed fuselet.

While the above are simple notional programs, they demonstrate how fuselet information needs can be specified using semantic web services, and its business logic specified using workflow process models. Moreover, by suitably restricting the domain of discourse to publish-subscribe events, workflow composition can be straightforwardly effected using Prolog-like theorem proving. Such backwards-reasoning from requirements to fuselet workflow can be used both dynamically to re-engineer fuselet compositions as well as statically to guide development of workflows that satisfy information on demand.

A.4 Demonstration Programs

A.4.1 OWL-S-Based Fuselet Specification.

A.4.1.1 Service Ontology.

The OWL ontology for the example selector predicates is shown below.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!--
#=========================================================================
=
# Copyright (C) 2005 Orielle, LLC.
#=========================================================================
=
-->

<!--
#=========================================================================
=
# The OWL-S service profile references two other fuselet-specific ontologies:
#  ProfileHierarchy - holds general ontology for fuselet processes
#  SelectorHierarchy - holds ontology for selector predicates
#    and manipulation functions.
#=========================================================================
=
-->

<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">

  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl">

  <!ENTITY DEFAULT "http://www.orielle.com/URI">
  <!ENTITY THIS    "FuseletService.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs  = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"

  xmlns:service = "&service;#"
  xmlns:profile = "&profile;#"
  xmlns:process = "&process;#"
  xmlns:grounding = "&grounding;#"

  xmlns       = "&DEFAULT;#"
  xml:base  = "&DEFAULT;"
 >
```

```
 <owl:Ontology rdf:about="">
   <owl:versionInfo> 3.0 </owl:versionInfo>
   <rdfs:comment>
     This is an ontology for Fuselet service hierarchy,
   and for selector hierarchy, both used for fuselet composition.
   </rdfs:comment>
   <owl:imports rdf:resource="&service;" />
   <owl:imports rdf:resource="&profile;" />
   <owl:imports rdf:resource="&process;" />
   <owl:imports rdf:resource="&grounding;" />
 </owl:Ontology>

<!-- ########################################################################## -->
<!-- # Fuselet class definitions                                    -->
<!-- ########################################################################## -->

  <owl:Class rdf:ID="Fuselet">
    <rdfs:subClassOf rdf:resource="&profile;#Profile" />
    <rdfs:comment>
   Class that represents all Fuselet services.
    </rdfs:comment>
  </owl:Class>

<!-- ########################################################################## -->
<!-- # Selector & manipulation function hierarchy, and JBI InfoObject types
-->
<!-- ########################################################################## -->

  <owl:Class rdf:ID="JBIInfoObject">
  </owl:Class>

  <owl:Class rdf:ID="JBISelector">
  </owl:Class>

  <owl:Class rdf:ID="sensorTopic">
    <rdfs:subClassOf rdf:resource="#JBIInfoObject"/>
  </owl:Class>

  <owl:Class rdf:ID="sensorTopic2">
    <rdfs:subClassOf rdf:resource="#JBIInfoObject"/>
  </owl:Class>

  <owl:Class rdf:ID="trackTopic">
    <rdfs:subClassOf rdf:resource="#JBIInfoObject"/>
  </owl:Class>

  <owl:Class rdf:ID="threatTopic">
    <rdfs:subClassOf rdf:resource="#JBIInfoObject"/>
  </owl:Class>

  <owl:Class rdf:ID="sensorSelector">
    <rdfs:subClassOf rdf:resource="#JBISelector"/>
  </owl:Class>
```

```
  <owl:Class rdf:ID="sensorSelector2">
    <rdfs:subClassOf rdf:resource="#JBISelector"/>
  </owl:Class>

  <owl:Class rdf:ID="trackSelector">
    <rdfs:subClassOf rdf:resource="#JBISelector"/>
  </owl:Class>

  <owl:Class rdf:ID="threatSelector">
    <rdfs:subClassOf rdf:resource="#JBISelector"/>
  </owl:Class>

  <owl:Class rdf:ID="refinedThreatSelector">
    <rdfs:subClassOf rdf:resource="#threatSelector"/>
  </owl:Class>

  <owl:Class rdf:ID="refinedTrackSelector">
    <rdfs:subClassOf rdf:resource="#trackSelector"/>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="track">
    <rdfs:domain rdf:resource="#JBIInfoObject"/>
    <rdfs:range rdf:resource="&rdf;#list"/>
  </owl:ObjectProperty>

</rdf:RDF>
```

Figure 8. Service Ontology with Selector Terms.

A.4.1.2 Tracker Fuselet Service Profile.

The OWL-S specification for a tracker fuselet is shown below. The OWL-S for the threat prediction fuselet and sensor publisher client are similar and are omitted for brevity. The tracking fuselet subscribes to sensor returns and publishes updated tracks, while the threat prediction fuselet subscribes to tracks and publishes threat warnings. The OWL-S specification illustrates the use of the Semantic Web Rule Language (SWRL) for expressing the Input/Output/Precondition/Effects. The grounding is included in the metadata to facilitate workflow composition.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!--
# Copyright (C) 2005 Orielle, LLC.
-->

<!--
#=============================================================================
=
# This document provides an OWL-S specification
# for a tracker fuselet service.
```

```
#
# Logic specification
# ===================
# The tracker service has a Horn clause specification of:
#    tracker_fuselet(Z) {
#  publish(trackTopic, refinedTrackSelector, track(X,Z)))
#    IF subscribe(sensorTopic, sensorSelector, X)
#
#  publish(trackTopic, trackSelector, track2(X,Y,Z)))
#    IF subscribe(sensorTopic, sensorSelector, X)
#    AND subscribe(sensorTopic2, sensorSelector2, Y)
#    }
# where uppercase words denote logical variables.
#
# We slightly alter the parameterization to match the generic
# BPEL fuselet template of <sphere, inTopic, inSelector, transform,
# outTopic, outSelector>.
#
# OWL-S Extensions:
#  We introduce a notion of default attribute for parameterization
# for demand-driven fuselet instantiation.
# We also add a "transform" attribute to indicate transform properties,
# and add a "name" attribute to SimpleProcess.
#
# OWL-S specification
# ===================
# The OWL-S specification consists of a:
# 1. A service outline with links to the profile, process, and grounding.
# 2. A service profile with links to inputs/outputs/preconditions/effects
#  and parameters encoded in the process model.
#  The profile references two other fuselet-specific ontologies:
#    ProfileHierarchy - holds general ontology for fuselet processes
#    SelectorHierarchy - holds ontology for selector predicates
#       and manipulation functions.
#  The profile is a variant of OWL-S in that it allows "hasParameter",
#  which is typically used only within inputs & outputs.
#  This is to allow parameterized fuselets (and parameterized services).
# 3. A process model consisting of:
#  SimpleProcess for the fuselet abstraction.
#  CompositeProcess to hold the BPEL.
#    The composite process basically resolves to:
#  process(parameters) {
#    foreach (result) { if inCondition => hasEffects ; withOutputs }
#  }
# 4. A grounding from OWL service parameters to WSDL.
#  (A notional grounding is shown here. The WSDL is omitted.)
#
# Rule specification
# ==================
# Preconditions and effects are expressed using
# RuleML (FOL) with SWRL extensions.
# An alternative is to use RuleML, or SWRL native.
# The RuleML conditions basically consist of a conjunction of atoms,
```

```
# while SWRL restricts atoms to binary predicates and
# unary class/data-range predicates.
#
# RuleML version is similar to the if-then rules above.
#
# Swrl version is:
#  publish(topic, Output) ^ class(Output, selectorOut)
#    ^ transform(Output, Subscriptions)
#    ^ swrlb:listConcat (Subscriptions, X, Y)
#  IF subscribe(topicIn1, X) ^ class(X, selectorIn1)
#  AND  subscribe(topicIn2, X) ^ class(X, selectorIn2)
#
# Service Profile
# ===============
# We treat the subscribe and publish operations as preconditions and effects,
# and set the inputs and ouputs to be the maximal covering of the
# publish and subscribe topics.
#
# Each of the "publish" clauses above is internally represented
# in the service profile as:
#
# Parameter: Z is_of_type String
# Input:  X  is_of_type sensorTopic
# Input:  Y  is_of_type sensorTopic2
# Output: T  is_of_type trackTopic
# // for each publish if-then clause
# hasResult {
#  incondition {
#    right-hand "if" side
#  }
#  hasEffect {
#    left-hand "then" side
#  }
# }
# hasResult ... // for each publish if-then clause
#
# where:  topics are subclasses of JBIInfoObject,
#  and selectors are classes arranged in a lattice
#  (or selectors could be made predicates).
#=============================================================================
=
-->

<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">

  <!ENTITY swrlx "http://www.w3.org/2003/11/swrlx">
  <!ENTITY ruleml "http://www.w3.org/2003/11/ruleml">
  <!ENTITY owlx "http://www.w3.org/2003/05/owl-xml">
  <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb">
```

```
  <!ENTITY expr "http://www.daml.org/services/owl-
s/1.1/generic/Expression.owl">
  <!ENTITY swrl "http://www.w3.org/2003/11/swrl">
  <!ENTITY objList "http://www.daml.org/services/owl-
s/1.1/generic/ObjectList.owl">
  <!ENTITY time "http://www.isi.edu/~pan/damltime/time-entry.owl">
  <!ENTITY actor "http://www.daml.org/services/owl-s/1.1/ActorDefault.owl">

  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl">

  <!ENTITY profileHierarchy "ProfileHierarchy.owl">
  <!ENTITY selectorHierarchy "SelectorHierarchy.owl">

  <!ENTITY fuselet_service "FuseletService.owl">
  <!ENTITY fuselet_profile "FuseletProfile.owl">
  <!ENTITY fuselet_process "FuseletProcess.owl">
  <!ENTITY fuselet_grounding "FuseletGrounding.owl">
  <!ENTITY fuselet_wsdl "FuseletGroundingWsdl.wsdl">

  <!ENTITY DEFAULT "FuseletService.owl">
  <!ENTITY THIS    "FuseletService.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs  = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:expr = "&expr;#"
  xmlns:swrl = "&swrl;#"
  xmlns:objList =    "&objList;#"

  xmlns:swrlx = "&swrlx;#"
  xmlns:ruleml = "&ruleml;#"
  xmlns:owlx = "&owlx;#"
  xmlns:swrlb = "&swrlb;#"

  xmlns:time  = "&time;#"
  xmlns:actor  = "&actor;#"

  xmlns:service = "&service;#"
  xmlns:profile = "&profile;#"
  xmlns:process = "&process;#"
  xmlns:grounding = "&grounding;#"

  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:selectorHierarchy = "&selectorHierarchy;#"

  xmlns:fuselet_service = "&fuselet_service;#"
```

```
  xmlns:fuselet_profile = "&fuselet_profile;#"
  xmlns:fuselet_process = "&fuselet_process;#"
  xmlns:fuselet_grounding = "&fuselet_grounding;#"
  xmlns:fuselet_wsdl = "&fuselet_wsdl;#"

  xmlns       = "&DEFAULT;#"
  xml:base  = "&DEFAULT;"
 >

 <owl:Ontology rdf:about="">
   <owl:versionInfo>
     $Id: spec.xml 1.1 2005/11/14 22:33:31Z phmills Development $
   </owl:versionInfo>
   <rdfs:comment>
   This ontology represents the OWL-S service description for the
   Fuselet Tracker web service example.
   </rdfs:comment>
   <owl:imports rdf:resource="&service;" />
   <owl:imports rdf:resource="&profile;" />
   <owl:imports rdf:resource="&process;" />
   <owl:imports rdf:resource="&grounding;" />
   <owl:imports rdf:resource="&time;" />
   <owl:imports rdf:resource="&actor;" />
   <owl:imports rdf:resource="&profileHierarchy;" />
   <owl:imports rdf:resource="&selectorHierarchy;" />
   <owl:imports rdf:resource="&fuselet_profile;" />
   <owl:imports rdf:resource="&fuselet_process;" />
   <owl:imports rdf:resource="&fuselet_grounding;" />
 </owl:Ontology>

<!--
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
#:::: FuseletService.owl
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
-->

<!-- ######################################################################## -->
<!-- # OWL-S links to service profile, process model, and grounding.      -->
<!-- ######################################################################## -->

<!--#####################################################
    # TrackerService
    #####################################################-->

<service:Service rdf:ID="TrackerService">
   <!-- Reference to the Profile -->
   <service:presents rdf:resource="&fuselet_profile;#TrackerProfile"/>

   <!-- Reference to the Process Model -->
   <service:describedBy rdf:resource="&fuselet_process;#TrackerProcess"/>
```

```
   <!-- Reference to the Grounding -->
   <service:supports rdf:resource="&fuselet_grounding;#TrackerGrounding"/>
</service:Service>

<!-- Inverse links -->

<profile:Profile rdf:about="&fuselet_profile;#TrackerProfile">
   <service:presentedBy rdf:resource="#TrackerService"/>
</profile:Profile>

<process:SimpleProcess rdf:about="&fuselet_process;#TrackerProcess">
   <service:describes rdf:resource="#TrackerService"/>
</process:SimpleProcess>

<grounding:WsdlGrounding rdf:about="&fuselet_grounding;#TrackerGrounding">
   <service:supportedBy rdf:resource="#TrackerService"/>
</grounding:WsdlGrounding>

<!--
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
#:::: FuseletProfile.owl
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
-->

<!-- ########################################################################## -->
<!-- # Profile: Instance Definition of Tracker Service                        -->
<!-- ########################################################################## -->

<!--
#============================================================================
=
# This defines the service profile for a Tracker service.
#
# The Parameters and Inputs/Outputs/Preconditions/Effects
# are links to the definitions in the simple process model.
#
# Tracker is defined as a Fuselet.  It inherits from
# the ontology of services, in ProfileHierarchy, that it is a Profile.
#
# As an alternative approach, Tracker could be specified outside the
# hierarchy of services by directly declaring it an instance of Profile,
# and adjusting the relevant properties accordingly.
#============================================================================
=
-->

<profileHierarchy:Fuselet
         rdf:ID="TrackerProfile">

    <!-- reference to the service specification -->
    <service:presentedBy rdf:resource="&fuselet_service;#TrackerService"/>
```

```
    <profile:has_process rdf:resource="&fuselet_process;#TrackerProcess"/>

    <profile:serviceName>Fuselet_Tracker_Service</profile:serviceName>
    <profile:textDescription>
        This service provides a tracker fuselet.
    </profile:textDescription>

    <profile:contactInformation>
      <actor:Actor rdf:ID="Tracker_contacts">
        <actor:name>Some Author</actor:name>
   <actor:title>JBI Info Managment Staff</actor:title>
        <actor:phone> N/A </actor:phone>
        <actor:fax> N/A </actor:fax>
        <actor:email> N/A </actor:email>
        <actor:physicalAddress> N/A </actor:physicalAddress>
        <actor:webURL> N/A </actor:webURL>
      </actor:Actor>
    </profile:contactInformation>

    <profile:hasParameter rdf:resource="&fuselet_process;#sphere"/>
    <profile:hasParameter rdf:resource="&fuselet_process;#inTopic"/>
    <profile:hasParameter rdf:resource="&fuselet_process;#inSelector"/>
    <profile:hasParameter rdf:resource="&fuselet_process;#transform"/>
    <profile:hasParameter rdf:resource="&fuselet_process;#outTopic"/>
    <profile:hasParameter rdf:resource="&fuselet_process;#outCondition"/>

    <profile:hasInput rdf:resource="&fuselet_process;#sensorTopicIn"/>
    <profile:hasInput rdf:resource="&fuselet_process;#sensorTopic2In"/>

    <profile:hasOutput rdf:resource="&fuselet_process;#trackTopicOut"/>

    <!--  No preconditions - These would be global "guards".
    <profile:hasPrecondition rdf:resource="&fuselet_process;#InputExists"/>
    -->

    <profile:hasResult rdf:resource="&fuselet_process;#publishClause1"/>
    <profile:hasResult rdf:resource="&fuselet_process;#publishClause2"/>

</profileHierarchy:Fuselet>

<!--
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
#:::: FuseletProcess.owl
#::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
-->

<!-- ########################################################################## -->
<!-- # Process Model: Atomic, Composite, and Simple Processes                 -->
<!-- ########################################################################## -->

<!--
```

```
#============================================================================
=
# This provides the process model for the service.
#
# The process model holds atomic as well as composite process definitions.
# In addition the process model holds simple process definitions, which are
# an abstract version of a composite process that is not executable.
#
# Two definitions are provided for the tracker service:
# a simple process definition which abstracts its IOPE,
# and a composite process definition which holds the BPEL.
#
# The process model also holds the definitions for
# the Parameters and Inputs/Outputs/Preconditions/Effects (IOPE),
# and uses SWRL (Semantic Web Rule Language) to encode the
# conditions and effects, which for fuselets are Horn clause if-then
statements.
#
# Selectors are modeled as classes (or data range predicates),
# with subtyping of selectors to capture information-object refinement.
# Transforms are represented as individual property predicates that hold
# between the output value and a list of other variables.
#============================================================================
=
-->

<!--#####################################################
    # Abstract simple process
    #####################################################-->

<process:SimpleProcess rdf:ID="TrackerProcess" name="trackFuselet">
  <process:expandsTo rdf:resource="#TrackerBPEL"/>

<!--#####################################################
    # Parameters
    #####################################################-->
<!-- Generic fuselet parameters : BPEL template supports subset of
     with fuselet specification in that it
     allows only one publish and subscribe.
     To be enhanced in follow-on work.
-->

  <process:hasParameter>
    <process:Parameter rdf:ID="sphere" default="jbi">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="inTopic" default="sensorTopic">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
```

```
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="inSelector" default="//metadata">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="transform" default="track">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="outTopic" default="trackTopic">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="outCondition" default="*">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

<!--#######################################################
    # Inputs and Outputs
    #######################################################-->
  <process:hasInput>
    <process:Input rdf:ID="sensorTopicIn">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&selectorHierarchy;#sensorTopic</process:paramete
rType>
    </process:Input>
  </process:hasInput>

  <process:hasInput>
    <process:Input rdf:ID="sensorTopic2In">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&selectorHierarchy;#sensorTopic2</process:paramet
erType>
    </process:Input>
  </process:hasInput>

  <process:hasOutput>
    <process:Output rdf:ID="trackTopicOut">
```

```
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&selectorHierarchy;#trackTopic</process:parameter
Type>
    </process:Output>
  </process:hasOutput>

<!--#######################################################
    # Conditions: An example of an empty precondition, or guard.
    #######################################################-->
  <process:hasPrecondition>
    <expr:SWRL-Condition rdf:ID="InputExists">
   <expr:expressionLanguage rdf:resource="&expr;#SWRL"/>
   <expr:expressionBody rdf:parseType="Literal">
     <swrl:AtomList rdf:about="&rdf;#nil"/>
   </expr:expressionBody>
    </expr:SWRL-Condition>
  </process:hasPrecondition>

<!--#######################################################
    # Effects: Result for each of the publish clauses.
    #######################################################-->
  <process:hasResult>
    <process:Result rdf:ID="publishClause1">
      <rdfs:comment>
             The first if-then clause of the fuselet publishing actions.
      </rdfs:comment>

      <process:inCondition>
          <expr:SWRL-Condition rdf:ID="">
            <expr:expressionBody rdf:parseType="Literal">

              <ruleml:And>
                <ruleml:And>
                  <swrlx:individualPropertyAtom  swrlx:property="subscribe">
                    <owlx:Individual owlx:name="sensorTopic" />
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:individualPropertyAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="sensorSelector" />
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:classAtom>
                </ruleml:And>
              </ruleml:And>

            </expr:expressionBody>
          </expr:SWRL-Condition>
      </process:inCondition>

      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#trackTopicOut"/>
     <!--
          <process:valueSource>
```

```
            <process:ValueOf>
              <process:theVar rdf:resource="#trackTopicOut"/>
            </process:ValueOf>
          </process:valueSource>
     -->
        </process:OutputBinding>
      </process:withOutput>

      <process:hasEffect>
        <expr:SWRL-Expression>
          <expr:expressionBody rdf:parseType="Literal">

            <ruleml:And>
              <ruleml:And>
                  <swrlx:individualPropertyAtom  swrlx:property="publish">
                    <owlx:Individual owlx:name="trackTopic" />
                    <ruleml:Var>Output</ruleml:Var>
                  </swrlx:individualPropertyAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="refinedTrackSelector" />
                    <ruleml:Var>Output</ruleml:Var>
                  </swrlx:classAtom>
                  <swrlx:individualPropertyAtom  swrlx:property="track"
transform="true">
                    <ruleml:Var>Output</ruleml:Var>
                    <ruleml:Var>Subscriptions</ruleml:Var>
                  </swrlx:individualPropertyAtom>
                  <swrlx:builtinAtom swrlx:builtin="&swrlb;#listConcat">
                    <ruleml:Var>Subscriptions</ruleml:Var>
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:builtinAtom>
              </ruleml:And>
            </ruleml:And>

          </expr:expressionBody>
        </expr:SWRL-Expression>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>

  <process:hasResult>
    <process:Result rdf:ID="publishClause2">
      <rdfs:comment>
             The second if-then clause of the fuselet publishing actions.
      </rdfs:comment>

      <process:inCondition>
          <expr:SWRL-Condition rdf:ID="">
            <expr:expressionBody rdf:parseType="Literal">

              <ruleml:And>
                <ruleml:And>
                  <swrlx:individualPropertyAtom  swrlx:property="subscribe">
```

```
                    <owlx:Individual owlx:name="sensorTopic" />
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:individualPropertyAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="sensorSelector" />
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:classAtom>
                </ruleml:And>
                <ruleml:And>
                  <swrlx:individualPropertyAtom  swrlx:property="subscribe">
                    <owlx:Individual owlx:name="sensorTopic2" />
                    <ruleml:Var>Y</ruleml:Var>
                  </swrlx:individualPropertyAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="sensorSelector2" />
                    <ruleml:Var>Y</ruleml:Var>
                  </swrlx:classAtom>
                </ruleml:And>
              </ruleml:And>

            </expr:expressionBody>
          </expr:SWRL-Condition>
      </process:inCondition>

      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#trackTopicOut"/>
     <!--
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource="#trackTopicOut"/>
            </process:ValueOf>
          </process:valueSource>
     -->
        </process:OutputBinding>
      </process:withOutput>

      <process:hasEffect>
        <expr:SWRL-Expression>
          <expr:expressionBody rdf:parseType="Literal">

            <ruleml:And>
              <ruleml:And>
                  <swrlx:individualPropertyAtom  swrlx:property="publish">
                    <owlx:Individual owlx:name="trackTopic" />
                    <ruleml:Var>Output</ruleml:Var>
                  </swrlx:individualPropertyAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="trackSelector" />
                    <ruleml:Var>Output</ruleml:Var>
                  </swrlx:classAtom>
                  <swrlx:individualPropertyAtom  swrlx:property="track2"
transform="true">
```

```
                    <ruleml:Var>Output</ruleml:Var>
                    <ruleml:Var>Subscriptions</ruleml:Var>
                  </swrlx:individualPropertyAtom>
                  <swrlx:builtinAtom swrlx:builtin="&swrlb;#listConcat">
                    <ruleml:Var>Subscriptions</ruleml:Var>
                    <ruleml:Var>X</ruleml:Var>
                    <ruleml:Var>Y</ruleml:Var>
                  </swrlx:builtinAtom>
              </ruleml:And>
            </ruleml:And>

          </expr:expressionBody>
        </expr:SWRL-Expression>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>

</process:SimpleProcess>

<!--#######################################################
    # BPEL (composite process)
    #######################################################-->

<process:CompositeProcess rdf:ID="TrackerBPEL">
  <rdfs:comment>
    This composite process variant holds the executable BPEL.
  </rdfs:comment>
  <process:collapsesTo rdf:resource="#TrackerProcess"/>
  <process:composedOf>
   <!-- BPEL goes here -->
  </process:composedOf>
</process:CompositeProcess>

<!--
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
#:::: FuseletGrounding.owl
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
-->

<!-- ######################################################################## -->
<!-- # Grounding:  Mapping of service inputs and outputs into WSDL.      -->
<!-- ######################################################################## -->

<!--#######################################################
    # Grounding for Tracker Service
    #
    #A real fuselet grounding will look substantially
    #   different, as it has start(), setParameter(),
    #   and other methods.
    #######################################################-->
```

```
<grounding:WsdlGrounding rdf:ID="TrackerGrounding">
  <rdfs:comment>
    The fuselet is an atomic process.
    There are also JBI atomic web services (not shown).
  </rdfs:comment>
  <grounding:hasAtomicProcessGrounding
rdf:resource="#TrackerAtomicGrounding"/>
</grounding:WsdlGrounding>

<!--#####################################################
    # Grounding for Atomic Process
    #####################################################-->

<grounding:WsdlAtomicProcessGrounding rdf:ID="TrackerAtomicGrounding">
  <grounding:owlsProcess rdf:resource="&fuselet_process;#TrackerProcess"/>
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#Tracker_PortType</grounding:portTy
pe>
      <grounding:operation
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#Tracker_operation</grounding:opera
tion>
    </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>

  <grounding:wsdlInputMessage
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#Tracker_Input</grounding:wsdlInput
Message>

  <grounding:wsdlInput>
    <grounding:WsdlInputMessageMap>
      <grounding:owlsParameter rdf:resource="&fuselet_process;#input1"/>
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#topicIn1</grounding:wsdlMessagePar
t>
    </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>

  <grounding:wsdlInput>
    <grounding:WsdlInputMessageMap>
      <grounding:owlsParameter rdf:resource="&fuselet_process;#input2"/>
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#topicIn2</grounding:wsdlMessagePar
t>
    </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>

  <grounding:wsdlOutputMessage
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#Tracker_Output</grounding:wsdlOutp
utMessage>

  <grounding:wsdlOutput>
```

```
    <grounding:WsdlOutputMessageMap>
      <grounding:owlsParameter rdf:resource="&fuselet_process;#output1"/>
      <grounding:wsdlMessagePart
rdf:datatype="&xsd;#anyURI">&fuselet_wsdl;#topicOut</grounding:wsdlMessagePar
t>
    </grounding:WsdlOutputMessageMap>
  </grounding:wsdlOutput>

  <grounding:wsdlReference rdf:datatype="&xsd;#anyURI">
   http://www.w3.org/TR/2001/NOTE-wsdl-20010315
  </grounding:wsdlReference>
  <grounding:wsdlDocument rdf:datatype="&xsd;#anyURI">
   &fuselet_wsdl;
  </grounding:wsdlDocument>
</grounding:WsdlAtomicProcessGrounding>

</rdf:RDF>
```

Figure 9. Tracker Fuselet - Service Profile.

A.4.2 BPEL-Based Fuselet Business Logic.

The WSDL and BPEL for the publish-subscribe CAPI implementation, and the WSDL and BPEL for the generic fuselet that can be parameterized for tracking and threat prediction fuselets as well as a sensor publisher client, are shown below.

A.4.2.1 Publish-Subscribe WSDL.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="PublishSubscribe"
        targetNamespace="http://com.orielle.bpel.publishSubscribe"
        xmlns:tns="http://com.orielle.bpel.publishSubscribe"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        >

<!-- CORRELATION PROPERTIES DEFINITION -->
    <bpws:property name="instanceId" type="xsd:string"/>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     TYPE DEFINITION - List of services participating in this BPEL process
     The default output of the BPEL designer uses strings as input and
     output to the BPEL Process. But you can define or import any XML
     Schema type and us them as part of the message types.
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <types>
        <schema attributeFormDefault="unqualified"
                elementFormDefault="qualified"
```

```
                targetNamespace="http://com.orielle.bpel.publishSubscribe"
                xmlns="http://www.w3.org/2001/XMLSchema"
                >

            <element name="initiateRequestType">
                <complexType>
                    <sequence>
                        <element name="params" type="string" />
                    </sequence>
                </complexType>
            </element>

            <element name="initiateResponseType">
                <complexType>
                    <sequence>
                        <element name="result" type="string"/>
                    </sequence>
                </complexType>
            </element>

            <complexType name="NameValuePairType">
               <sequence>
                   <element name="name" type="string"/>
                   <element name="value" type="string"/>
              </sequence>
            </complexType>

       <element name="PropertiesType">
                <complexType>
                  <sequence>
                      <element name="property" type="tns:NameValuePairType"
maxOccurs="unbounded"/>
                  </sequence>
                </complexType>
       </element>


       <element name="Sphere" type="string"/>
       <element name="Topic" type="string"/>
       <element name="Selector" type="string"/>
       <element name="MessageId" type="string"/>
       <element name="MessageType" type="string"/>
       <element name="Headers" type="string"/>
       <element name="Metadata" type="string"/>
       <element name="Payload" type="anyType"/>
       <element name="SessionId" type="string"/>
       <element name="Status" type="string"/>


        </schema>
    </types>
```

```
<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     MESSAGE TYPE DEFINITION - Definition of the message types used as
     part of the port type defintions
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <message name="messageEnvelope">
        <part name="metadata" element="tns:Metadata"/>
        <part name="payload" element="tns:Payload"/>
        <part name="correlationId" element="tns:SessionId"/>
    </message>

    <message name="subscriptionMessage">
        <part name="sphere" element="tns:Sphere"/>
        <part name="topic" element="tns:Topic"/>
        <part name="selector" element="tns:Selector"/>
    </message>

    <message name="sessionIdMessage">
        <part name="correlationId" element="tns:SessionId"/>
    </message>

    <message name="unsubscriptionMessage">
        <part name="correlationId" element="tns:SessionId"/>
    </message>

    <message name="initializeServerMessage">
        <part name="sphere" element="tns:Sphere"/>
    </message>

    <message name="statusMessage">
        <part name="status" element="tns:Status"/>
    </message>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     PORT TYPE DEFINITION - A port type groups a set of operations into
     a logical service unit.
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <!-- portType implemented by the PublishSubscribe BPEL process -->
    <portType name="PublishSubscribe">
        <operation name="initiateServer">
            <input name="initiateRequest"
message="tns:initializeServerMessage"/>
            <output name="initiateResponse" message="tns:statusMessage"/>
        </operation>

        <operation name="createPublisher">
            <input name="createPublisherRequest"
message="tns:subscriptionMessage"/>
            <output name="createPublisherResponse"
message="tns:sessionIdMessage"/>
        </operation>
```

```
        <operation name="sendMessage">
            <input name="sendMessageRequest" message="tns:messageEnvelope"/>
            <output name="sendMessageResponse" message="tns:statusMessage"/>
        </operation>


        <operation name="createSubscriber">
             <input name="createSubscriberRequest"
message="tns:subscriptionMessage"/>
        <output name="createSubscriberResponse"
message="tns:sessionIdMessage"/>
        </operation>

        <operation name="subscribe">
             <input name="subscribeRequest" message="tns:sessionIdMessage"/>
        <!--
             <input name="subscribeRequest"
message="tns:subscriptionMessage"/>
        -->
        <!--
        <output name="subscribeResponse" message="tns:sessionIdMessage"/>
        -->
        </operation>

        <operation name="getMessage">
            <input name="getMessageRequest" message="tns:sessionIdMessage"/>
            <output name="getMessageResponse" message="tns:messageEnvelope"/>
        </operation>

        <operation name="unsubscribe">
             <input name="unsubscribeRequest"
message="tns:unsubscriptionMessage"/>
        <output name="unsubscribeResponse" message="tns:statusMessage"/>
        </operation>
    </portType>

    <!-- portType implemented by the requester of PublishSubscribe BPEL
process
         for asynchronous callback purposes
         -->
    <portType name="SubscribeCallback">
        <operation name="onMessage">
            <input message="tns:messageEnvelope"/>
        </operation>
    </portType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     PARTNER LINK TYPE DEFINITION
         the PublishSubscribe partnerLinkType binds the provider and
         requester portType into an asynchronous conversation.
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <plnk:partnerLinkType name="PublishSubscribe">
```

```
        <plnk:role name="PublishSubscribeProvider">
            <plnk:portType name="tns:PublishSubscribe"/>
        </plnk:role>
        <plnk:role name="PublishSubscribeRequester">
            <plnk:portType name="tns:SubscribeCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>

<!-- CORRELATION PROPERTIES DEFINITION -->
    <bpws:propertyAlias propertyName="tns:instanceId"
        messageType="tns:messageEnvelope" part="correlationId"
        query="/tns:SessionId" />

    <bpws:propertyAlias propertyName="tns:instanceId"
        messageType="tns:sessionIdMessage" part="correlationId"
          query="/tns:SessionId" />
</definitions>
```

Figure 10. Publish Subscribe - WSDL Interface Definition.

A.4.2.2 Publish-Subscribe BPEL implementation.

```
<!-- PublishSubscribe BPEL Process [Generated by the Oracle BPEL Designer]  -
->
<process name="PublishSubscribe"
targetNamespace="http://com.orielle.bpel.publishSubscribe"
suppressJoinFailure="yes"
xmlns:tns="http://com.orielle.bpel.publishSubscribe"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns:ora="http://schemas.oracle.com/xpath/extension">
   <bpelx:exec import="java.util.*"/>
   <bpelx:exec import="java.lang.*"/>
   <bpelx:exec import="java.rmi.RemoteException"/>
   <bpelx:exec import="org.w3c.dom.Element"/>
   <bpelx:exec import="com.orielle.saffire.Extensions.JBI.*"/>
   <bpelx:exec import="com.orielle.saffire.Extensions.XmlSpaces.*"/>
   <bpelx:exec import="org.infospherics.commonAPI.impl.*"/>
   <!-- ================================================================= -->
   <!-- PARTNERLINKS                                                      -->
   <!-- List of services participating in this BPEL process               -->
   <!-- ================================================================= -->
   <partnerLinks>
     <!--
        The 'client' role represents the requester of this service. It is
        used for callback. The location and correlation information
associated
        with the client role are automatically set using WS-Addressing.
        -->
     <partnerLink name="client" partnerLinkType="tns:PublishSubscribe"
myRole="PublishSubscribeProvider" partnerRole="PublishSubscribeRequester"/>
   </partnerLinks>
   <!-- ================================================================= -->
```

```
   <!-- VARIABLES                                                          -->
   <!-- List of messages and XML documents used within this BPEL process  -->
   <!-- ================================================================= -->
   <variables>
     <!-- Reference to the message passed as input during initiation -->
     <variable name="subscriptionRequest"
messageType="tns:subscriptionMessage"/>
     <variable name="message" messageType="tns:messageEnvelope"/>
     <variable messageType="tns:initializeServerMessage" name="sphere"/>
     <!-- Reference to the message that will be sent back to the
             requester during callback
             -->
     <variable name="correlationId" messageType="tns:sessionIdMessage"/>
     <variable name="status" messageType="tns:statusMessage"/>
   </variables>
   <!-- Correlation sets -->
   <correlationSets>
     <correlationSet name="Instance_set" properties="tns:instanceId"/>
   </correlationSets>
   <!-- ================================================================= -->
   <!-- ORCHESTRATION LOGIC                                               -->
   <!-- Set of activities coordinating the flow of messages across the    -->
   <!-- services integrated within this business process                  -->
   <!-- ================================================================= -->
   <sequence name="main">
     <!-- Receive input from requestor.
             Note: This maps to operation defined in PublishSubscribe.wsdl
             -->
     <pick name="pick-1" createInstance="yes">
       <onMessage partnerLink="client" portType="tns:PublishSubscribe"
operation="createPublisher" variable="subscriptionRequest">
         <sequence>
           <assign name="assign-1">
             <copy>
               <from expression="ora:getInstanceId()"></from>
               <to variable="correlationId" part="correlationId"
query="/tns:SessionId"/>
             </copy>
           </assign>
           <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="exec-1">
             <![CDATA[
//============================================================================
=
// Java code snippet goes here ... : createPublisher
//============================================================================
=
   // Retrieve variables from scope
   Element ssn = (Element) getVariableData("correlationId", "correlationId",
"/SessionId");
   String correlationId = ssn.getNodeValue();
   ssn = (Element) getVariableData("subscriptionRequest", "topic", "/Topic");
```

```
   String topic = ssn.getNodeValue();

   // Create connection and publisher
   try {

   ConnectionManager manager = new JbiConnectionManager();
   Connection connection = manager.createConnection(correlationId,
"clientDescription");
   PublisherSequence publisher = connection.createPublisherSequence(topic,
"1.0");

   } catch (Exception e) { addAuditTrailEntry("CreatePublisher error", e); };
//============================================================================
=
              ]]>
            </bpelx:exec>
            <reply name="reply-1" partnerLink="client"
portType="tns:PublishSubscribe" operation="createPublisher"
variable="correlationId"/>
          </sequence>
        </onMessage>
        <onMessage partnerLink="client" portType="tns:PublishSubscribe"
operation="subscribe" variable="correlationId">
          <sequence name="sequence-1">
            <scope name="scope-2">
              <variables>
                <variable messageType="tns:unsubscriptionMessage"
name="terminationId"/>
              </variables>
              <eventHandlers>
                <onMessage partnerLink="client"
portType="tns:PublishSubscribe" operation="unsubscribe"
variable="terminationId">
                  <sequence>
                    <reply name="reply-1" partnerLink="client"
portType="tns:PublishSubscribe" operation="unsubscribe" variable="status"/>
                    <terminate name="terminate-2"/>
                  </sequence>
                </onMessage>
              </eventHandlers>
              <while name="while-2" condition="true()">
                <sequence>
                  <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="exec-4">
                    <![CDATA[
//============================================================================
=
// Java code snippet goes here ... subscribe callback
//============================================================================
=
   // Retrieve variables from scope
```

```
   Element ssn = (Element) getVariableData("correlationId", "correlationId",
"/SessionId");
   String correlationId = ssn.getNodeValue();

   // Lookup connection and subscriber
   try {

   ConnectionManager manager = new JbiConnectionManager();
   Connection connection = manager.getConnection(correlationId);
   if (connection == null) {
     addAuditTrailEntry("No connection to subscriber");
   } else {
     addAuditTrailEntry("Connection to subscriber");
   }
   SubscriberSequence[] subscribers = connection.getSubscriberSequences();
   if (subscribers.length == 0) {
     addAuditTrailEntry("No subscribers");
   } else {
     addAuditTrailEntry("Found subscribers");
   }
   SubscriberSequence subscriber = subscribers[0];

   // Get message -- blocking wait (unlike JBI)
   InfoObject[] infoObjects = subscriber.getInfoObject();

   InfoObject infoObj = infoObjects[0];
   String metadata = infoObj.getInfoObjectMetadata();
   byte[] contentByte = (byte[]) infoObj.getInfoObjectPayload();
   String content = new String(contentByte);

   // Set result variables
   setVariableData("message", "metadata", "/Metadata", metadata);
   setVariableData("message", "payload", "/Payload", content);
   setVariableData("message", "correlationId", "/SessionId", correlationId);

   } catch (Exception e) { addAuditTrailEntry("Subscribe error", e); };
//=========================================================================
=
                     ]]>
                   </bpelx:exec>
                   <invoke name="invoke-2" partnerLink="client"
portType="tns:SubscribeCallback" operation="onMessage"
inputVariable="message"/>
                 </sequence>
               </while>
             </scope>
           </sequence>
         </onMessage>
         <onMessage partnerLink="client" portType="tns:PublishSubscribe"
operation="initiateServer" variable="sphere">
           <sequence name="sequence-1">
```

```
            <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="exec-1">
              <![CDATA[
//============================================================================
=
// Java code snippet goes here ... initiateServer
//============================================================================
=
   // Retrieve variables from scope
   Element ssn = (Element) getVariableData("sphere", "sphere", "/Sphere");
   String sphere = ssn.getNodeValue();

   //===== Create space
   try {

   IXmlSpaceManager manager = new XmlSpaceManager();
   // String sphere = "jbi.pubSub";
   manager.createSpace(sphere);
   JbiConnectionManager.setSphere(sphere);

   //===== Start the server
   Properties properties = new Properties();
   manager.createServer(sphere, properties);
   ISpaceServer server = manager.getServer(sphere);
   server.start();

   } catch (Exception e) { addAuditTrailEntry("Server initialization error",
e); };
//============================================================================
=
              ]]>
            </bpelx:exec>
            <reply name="reply-1" partnerLink="client"
portType="tns:PublishSubscribe" operation="initiateServer"
variable="status"/>
          </sequence>
        </onMessage>
        <onMessage partnerLink="client" portType="tns:PublishSubscribe"
operation="sendMessage" variable="message">
          <sequence>
            <assign name="assign-2">
              <copy>
                <from variable="message" part="correlationId"
query="/tns:SessionId"></from>
                <to variable="correlationId" part="correlationId"
query="/tns:SessionId"/>
              </copy>
            </assign>
            <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="exec-3">
              <![CDATA[
```

```
//============================================================================
=
// Java code snippet goes here ... publisher sendMessage
//============================================================================
=
   // Retrieve variables from scope
   Element ssn = (Element) getVariableData("correlationId", "correlationId",
"/SessionId");
   String correlationId = ssn.getNodeValue();

   // Get message variables
   ssn = (Element) getVariableData("message", "metadata", "/Metadata");
   String metadata = ssn.getNodeValue();
   ssn = (Element) getVariableData("message", "payload", "/Payload");
   String content = ssn.getNodeValue();

   // Lookup connection and subscriber
   try {

   ConnectionManager manager = new JbiConnectionManager();

   Connection connection = manager.getConnection(correlationId);
   if (connection == null) {
     addAuditTrailEntry("No connection to publisher");
   } else {
     addAuditTrailEntry("Connection to publisher");
   }
   PublisherSequence[] publishers = connection.getPublisherSequences();
   if (publishers.length == 0) {
     addAuditTrailEntry("No publishers");
   } else {
     addAuditTrailEntry("Found publisher");
   }
   PublisherSequence publisher = publishers[0];


   // Put message
   String messageType = "jbi";

   if (! metadata.startsWith("<")) {      // attach XML element
     metadata = "<metadata>" + metadata + "</metadata>";
   };
   if (! metadata.startsWith("<?xml")) {  // prepend XML prolog
     metadata = "<?xml version='1.0' encoding='ISO-8859-1'?>"
       + metadata;
   };

   System.out.println("Published metadata: " + metadata);

   InfoObject infoObject = connection.createInfoObject(messageType, "1.0",
content, metadata);
   publisher.publishInfoObject(infoObject);
```

```
   } catch (Exception e) { addAuditTrailEntry("Publish error", e); };
//============================================================================
=
                     ]]>
            </bpelx:exec>
            <reply name="reply-2" partnerLink="client"
portType="tns:PublishSubscribe" operation="sendMessage" variable="status"/>
          </sequence>
        </onMessage>
        <onMessage partnerLink="client" portType="tns:PublishSubscribe"
operation="createSubscriber" variable="subscriptionRequest">
          <sequence xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
process/">
            <assign name="assign-2">
              <copy>
                <from expression="ora:getInstanceId()"></from>
                <to variable="correlationId" part="correlationId"
query="/tns:SessionId"/>
              </copy>
            </assign>
            <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="exec-2">
              <![CDATA[
//============================================================================
=
// Java code snippet goes here ... createSubscriber
//============================================================================
=
   // Retrieve variables from scope
   Element ssn = (Element) getVariableData("correlationId", "correlationId",
"/SessionId");
   String correlationId = ssn.getNodeValue();
   ssn = (Element) getVariableData("subscriptionRequest", "topic", "/Topic");
   String topic = ssn.getNodeValue();
   ssn = (Element) getVariableData("subscriptionRequest", "selector",
"/Selector");
   String selector = ssn.getNodeValue();

   // Create connection and subscriber
   try {

   ConnectionManager manager = new JbiConnectionManager();
   Connection connection = manager.createConnection (correlationId,
"clientDescription");
   SubscriberSequence subscriber = connection.createSubscriberSequence(topic,
"1.0");
   subscriber.setSequencePredicate(selector);
   subscriber.setSequenceAttribute("warn", "1.0", "true");
   subscriber.activateSequence();

   } catch (Exception e) { addAuditTrailEntry("CreateSubscriber error", e);
};
```

```
//=============================================================================
=
              ]]>
            </bpelx:exec>
            <reply name="reply-1" partnerLink="client"
portType="tns:PublishSubscribe" operation="createSubscriber"
variable="correlationId"/>
          </sequence>
        </onMessage>
        <onMessage partnerLink="client" portType="tns:PublishSubscribe"
operation="getMessage" variable="correlationId">
          <sequence>
            <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="exec-4">
              <![CDATA[
//=============================================================================
=
// Java code snippet goes here ... getMessage
//=============================================================================
=
   // Retrieve variables from scope
   Element ssn = (Element) getVariableData("correlationId", "correlationId",
"/SessionId");
   String correlationId = ssn.getNodeValue();

   // Lookup connection and subscriber
   try {

   ConnectionManager manager = new JbiConnectionManager();
   Connection connection = manager.getConnection(correlationId);
   if (connection == null) {
     addAuditTrailEntry("No connection to subscriber");
   } else {
     addAuditTrailEntry("Connection to subscriber");
   }
   SubscriberSequence[] subscribers = connection.getSubscriberSequences();
   if (subscribers.length == 0) {
     addAuditTrailEntry("No subscribers");
   } else {
     addAuditTrailEntry("Found subscribers");
   }
   SubscriberSequence subscriber = subscribers[0];

   // Get message -- blocking wait (unlike JBI)
   InfoObject[] infoObjects = subscriber.getInfoObject();

   InfoObject infoObj = infoObjects[0];
   String metadata = infoObj.getInfoObjectMetadata();
   byte[] contentByte = (byte[]) infoObj.getInfoObjectPayload();
   String content = new String(contentByte);

   // Set result variables
```

```
   setVariableData("message", "metadata", "/Metadata", metadata);
   setVariableData("message", "payload", "/Payload", content);
   setVariableData("message", "correlationId", "/SessionId", correlationId);

   } catch (Exception e) { addAuditTrailEntry("Subscribe error", e); };
//=============================================================================
=
                     ]]>
            </bpelx:exec>
            <reply name="reply-1" partnerLink="client"
portType="tns:PublishSubscribe" operation="getMessage" variable="message"/>
          </sequence>
        </onMessage>
     </pick>
     <!-- Generate reply to synchronous request -->
     <!-- Asynchronous callback to the requester.
             Note: the callback location and correlation id is transparently
handled
             using WS-addressing.
             -->
     <!--
        <invoke name="callbackClient"
                partnerLink="client"
                portType="tns:SubscribeCallback"
                operation="onMessage"
                inputVariable="message"
                />
        -->
   </sequence>
</process>
```

Figure 11. Publish Subscribe - BPEL Business Logic.

A.4.2.3 Generic Fuselet WSDL.

```
<?xml version="1.0"?>
<definitions name="FuseletTransform"
        targetNamespace="http://com.orielle.bpel.fuselets"
        xmlns:tns="http://com.orielle.bpel.fuselets"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        >

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     TYPE DEFINITION - List of services participating in this BPEL process
     The default output of the BPEL designer uses strings as input and
     output to the BPEL Process. But you can define or import any XML
     Schema type and us them as part of the message types.
```

```
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <types>
        <schema attributeFormDefault="unqualified"
                elementFormDefault="qualified"
                targetNamespace="http://com.orielle.bpel.fuselets"
                xmlns="http://www.w3.org/2001/XMLSchema"
                >

            <element name="Parameters">
                <complexType>
                    <sequence>
                        <element name="sphere" type="string" />
                        <element name="inTopic" type="string" />
          <!-- none -->
                        <element name="inSelector" type="string" />
          <!-- none -->
                        <element name="transform" type="string" />
          <!-- sensor, track, receiver -->
                        <element name="outTopic" type="string" />
                        <element name="outCondition" type="string" />
                    </sequence>
                </complexType>
            </element>

            <element name="InstanceId">
                <complexType>
                    <sequence>
                        <element name="instanceId" type="string"/>
                    </sequence>
                </complexType>
            </element>

       <element name="SessionId" type="string"/>

        </schema>
    </types>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     MESSAGE TYPE DEFINITION - Definition of the message types used as
     part of the port type defintions
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <message name="inputParametersMessage">
        <part name="payload" element="tns:Parameters"/>
    </message>

    <message name="instanceIdMessage">
        <part name="payload" element="tns:InstanceId"/>
    </message>

    <message name="sessionIdMessage">
        <part name="correlationId" element="tns:SessionId"/>
```

```
    </message>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     PORT TYPE DEFINITION - A port type groups a set of operations into
     a logical service unit.
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <!-- portType implemented by the FuseletTransform BPEL process -->

    <portType name="FuseletTransform">
        <operation name="initiate">
            <input name="initiateRequest"
message="tns:inputParametersMessage"/>
            <output name="initiateResponse" message="tns:instanceIdMessage"/>
   </operation>

   <operation name="start">
       <input name="startRequest" message="tns:sessionIdMessage"/>
   </operation>

   <operation name="pause">
       <input name="pauseRequest" message="tns:sessionIdMessage"/>
   </operation>

   <operation name="resume">
       <input name="resumeRequest" message="tns:sessionIdMessage"/>
   </operation>

   <operation name="stop">
       <input name="stopRequest" message="tns:sessionIdMessage"/>
   </operation>
    </portType>

    <!-- portType implemented by the requester of FuseletTransform BPEL
process
         for asynchronous callback purposes
         -->
    <portType name="FuseletTransformCallback">
        <operation name="onResult">
            <input message="tns:instanceIdMessage"/>
        </operation>
    </portType>

<!-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
     PARTNER LINK TYPE DEFINITION
         the FuseletTransform partnerLinkType binds the provider and
         requester portType into an asynchronous conversation.
     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ --
>
    <plnk:partnerLinkType name="FuseletTransform">
        <plnk:role name="FuseletTransformProvider">
            <plnk:portType name="tns:FuseletTransform"/>
        </plnk:role>
```

```
        <plnk:role name="FuseletTransformRequester">
            <plnk:portType name="tns:FuseletTransformCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Figure 12. Generic Fuselet - WSDL Interface Definition.

A.4.2.4 Generic Fuselet BPEL implementation.

```
<!-- FuseletTransform BPEL Process [Generated by the Oracle BPEL Designer]  -
->
<process name="FuseletTransform"
targetNamespace="http://com.orielle.bpel.fuselets" suppressJoinFailure="yes"
xmlns:tns="http://com.orielle.bpel.fuselets"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:jbi="http://com.orielle.bpel.publishSubscribe"
xmlns:nsxml0="http://schemas.xmlsoap.org/ws/2003/03/addressing"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
   <bpelx:exec import="java.util.*"/>
   <bpelx:exec import="java.lang.*"/>
   <bpelx:exec import="java.rmi.RemoteException"/>
   <bpelx:exec import="org.w3c.dom.Element"/>
   <bpelx:exec import="com.orielle.saffire.Extensions.JBI.*"/>
   <bpelx:exec import="com.orielle.saffire.Extensions.XmlSpaces.*"/>
   <bpelx:exec import="org.infospherics.commonAPI.impl.*"/>
   <!-- ================================================================= -->
   <!-- PARTNERLINKS                                                      -->
   <!-- List of services participating in this BPEL process               -->
   <!-- ================================================================= -->
   <partnerLinks>
     <!--
        The 'client' role represents the requester of this service. It is
        used for callback. The location and correlation information
associated
        with the client role are automatically set using WS-Addressing.
        -->
     <partnerLink name="client" partnerLinkType="tns:FuseletTransform"
myRole="FuseletTransformProvider" partnerRole="FuseletTransformRequester"/>
     <partnerLink name="subscribeService"
partnerLinkType="jbi:PublishSubscribe" partnerRole="PublishSubscribeProvider"
myRole="PublishSubscribeRequester"/>
     <partnerLink name="publishService"
partnerLinkType="jbi:PublishSubscribe" partnerRole="PublishSubscribeProvider"
myRole="PublishSubscribeRequester"/>
   </partnerLinks>
   <!-- ================================================================= -->
   <!-- VARIABLES                                                         -->
   <!-- List of messages and XML documents used within this BPEL process  -->
   <!-- ================================================================= -->
```

```
   <variables>
     <!-- Reference to the message passed as input during initiation -->
     <variable name="params" messageType="tns:inputParametersMessage"/>
     <!-- Reference to the message that will be sent back to the
            requester during callback
            -->
     <variable name="instanceId" messageType="tns:instanceIdMessage"/>
     <variable name="inSessionId" messageType="tns:sessionIdMessage"/>
     <variable name="isPaused" type="xsd:boolean"/>
     <variable name="subscriptionRequest"
messageType="jbi:subscriptionMessage"/>
     <variable name="inMessage" messageType="jbi:messageEnvelope"/>
     <variable name="publisherCorrelationId"
messageType="jbi:sessionIdMessage"/>
     <variable name="publishRequest" messageType="jbi:subscriptionMessage"/>
     <variable name="outMessage" messageType="jbi:messageEnvelope"/>
     <variable name="subscriberCorrelationId"
messageType="jbi:sessionIdMessage"/>
     <variable name="status" messageType="jbi:statusMessage"/>
   </variables>
   <!-- ================================================================= -->
   <!-- ORCHESTRATION LOGIC                                               -->
   <!-- Set of activities coordinating the flow of messages across the    -->
   <!-- services integrated within this business process                  -->
   <!-- ================================================================= -->
   <sequence name="main">
     <!-- Receive input from requestor.
            Note: This maps to operation defined in FuseletTransform.wsdl
            -->
     <receive name="receiveInput" partnerLink="client"
portType="tns:FuseletTransform" operation="initiate" variable="params"
createInstance="yes"/>
     <!-- Asynchronous callback to the requester.
            Note: the callback location and correlation id is transparently
handled
            using WS-addressing.
            -->
     <assign name="setInstanceId">
       <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-1"/></bpeld:annotation><copy>
         <from expression="ora:getInstanceId()"></from>
         <to variable="instanceId" part="payload"
query="/tns:InstanceId/tns:instanceId"/>
       </copy>
       <copy>
         <from expression="false()"></from>
         <to variable="isPaused"/>
       </copy>
     </assign>
     <reply name="reply-1" partnerLink="client"
portType="tns:FuseletTransform" operation="initiate" variable="instanceId"/>
     <sequence name="TransformProcess">
```

```
        <assign name="setSubscriptionRequest">
          <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-2"/></bpeld:annotation><copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:sphere"></from>
            <to variable="publishRequest" part="sphere" query="/jbi:Sphere"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:sphere"></from>
            <to variable="subscriptionRequest" part="sphere"
query="/jbi:Sphere"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:inTopic"></from>
            <to variable="subscriptionRequest" part="topic"
query="/jbi:Topic"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:inSelector"></from>
            <to variable="subscriptionRequest" part="selector"
query="/jbi:Selector"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:outTopic"></from>
            <to variable="publishRequest" part="topic" query="/jbi:Topic"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:inSelector"></from>
            <to variable="publishRequest" part="selector"
query="/jbi:Selector"/>
          </copy>
        </assign>
        <switch name="switch-2">
          <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="switch-2"/></bpeld:annotation><case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:outTopic')) != 'none'">
            <invoke name="invoke-2" partnerLink="publishService"
portType="jbi:PublishSubscribe" operation="createPublisher"
inputVariable="publishRequest" outputVariable="publisherCorrelationId"/>
          </case>
          <otherwise>
            <empty name="empty-3"/>
          </otherwise>
        </switch>
        <switch name="switch-3">
```

```
          <case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:inTopic')) != 'none'">
            <invoke name="invoke-1" partnerLink="subscribeService"
portType="jbi:PublishSubscribe" operation="createSubscriber"
inputVariable="subscriptionRequest"
outputVariable="subscriberCorrelationId"/>
          </case>
          <otherwise>
            <empty name="empty-2"/>
          </otherwise>
        </switch>
        <scope name="InteractionHandler"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
          <eventHandlers>
            <onMessage partnerLink="client" portType="tns:FuseletTransform"
operation="stop" variable="inSessionId">
              <terminate name="terminate-1"/>
            </onMessage>
            <onMessage partnerLink="client" portType="tns:FuseletTransform"
operation="resume" variable="inSessionId">
              <empty/>
            </onMessage>
            <onMessage partnerLink="client" portType="tns:FuseletTransform"
operation="pause" variable="inSessionId">
              <assign name="setIsPaused">
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-4"/></bpeld:annotation><copy>
                  <from expression="true()"></from>
                  <to variable="isPaused"/>
                </copy>
              </assign>
            </onMessage>
          </eventHandlers>
          <while name="TransformLoop" condition="true()">
            <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="while-1"/></bpeld:annotation><sequence>
              <switch name="switch-4">
                <case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:inTopic')) != 'none'">
                  <sequence>
                    <invoke name="invoke-1" partnerLink="subscribeService"
portType="jbi:PublishSubscribe" operation="getMessage"
inputVariable="subscriberCorrelationId" outputVariable="inMessage"/>
                  </sequence>
                </case>
                <otherwise>
                </otherwise>
              </switch>
              <switch name="switch-1">
```

```
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="switch-1"/></bpeld:annotation><case
condition="boolean(bpws:getVariableData('isPaused'))">
                  <receive createInstance="no" name="receive-2"
partnerLink="client" portType="tns:FuseletTransform" operation="resume"
variable="inSessionId"/>
                </case>
                <otherwise>
                  <empty name="empty-1"/>
                </otherwise>
              </switch>
              <assign name="copyMessage">
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-2"/></bpeld:annotation><copy>
                  <from variable="inMessage"></from>
                  <to variable="outMessage"/>
                </copy>
              </assign>
              <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="transform">
                <![CDATA[
//============================================================================
=
// Java code snippet goes here ... perform transform
//============================================================================
=
   try {

   // Retrieve variables from scope
   Element ssn = (Element) getVariableData("publisherCorrelationId",
"correlationId", "/SessionId");
   String correlationId = ssn.getNodeValue();
   ssn = (Element)
getVariableData("params","payload","/tns:Parameters/tns:inTopic");
   String inTopic = ssn.getNodeValue();
   ssn = (Element)
getVariableData("params","payload","/tns:Parameters/tns:outTopic");
   String outTopic = ssn.getNodeValue();
   ssn = (Element)
getVariableData("params","payload","/tns:Parameters/tns:transform");
   String transform = ssn.getNodeValue();
   ssn = (Element) getVariableData("inMessage", "metadata", "/Metadata");
   String metadata = ssn.getNodeValue();
   ssn = (Element) getVariableData("inMessage", "payload", "/Payload");
   String content = ssn.getNodeValue();

   // Sleep for a while
   Thread.sleep(1000);

   // Perform transform
```

```
   //====
   // if (! inTopic.equals("none")) { };
   //====
   if (transform.equals("sensor")) {
     System.out.println("Sensor output generated");
     metadata = "<metadata><sensor>ewr</sensor></metadata>";
     content = "<sensorReturn>0 1 0</sensorReturn>";
   } else if (transform.equals("track")) {
     System.out.println("Track updated");
     metadata = "<metadata><track>ewr</track></metadata>";
     content = "<track>1 1 1</track>";
   } else if (transform.equals("threat")) {
     System.out.println("Threat issued");
     metadata = "<metadata><threat>ewr</threat></metadata>";
     content = "<threat>1 1 1</threat>";
   };
   //====
   // if (! outTopic.equals("none")) { };
   //====

   // Set result variables
   setVariableData("outMessage", "metadata", "/Metadata", metadata);
   setVariableData("outMessage", "payload", "/Payload", content);
   setVariableData("outMessage", "correlationId", "/SessionId",
correlationId);

   } catch (Exception e) { addAuditTrailEntry("Transform error", e); };
//=============================================================================
=
                ]]>
              </bpelx:exec>
              <assign name="setPubCorrelationId">
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-1"/></bpeld:annotation><copy>
                  <from variable="publisherCorrelationId"
part="correlationId" query="/jbi:SessionId"></from>
                  <to variable="outMessage" part="correlationId"
query="/jbi:SessionId"/>
                </copy>
              </assign>
              <switch name="switch-5">
                <case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:outTopic')) != 'none'">
                  <invoke name="invoke-3" partnerLink="publishService"
portType="jbi:PublishSubscribe" operation="sendMessage"
inputVariable="outMessage" outputVariable="status"/>
                </case>
                <otherwise>
                </otherwise>
              </switch>
            </sequence>
```

```
          </while>
        </scope>
      </sequence>
   </sequence>
</process>
```

Figure 13. Generic Fuselet - BPEL Business Logic.

A.4.3 Workflow Composition.

A.4.3.1 Specify Fuselet Goal.

The OWL-S for the goal of the fuselet to be synthesized is shown below. The fuselet goal is to subscribe to refined threat warnings. Instead of using OWL-S below, the goal might be alternatively described in shorter form as a Prova goal.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!--
# Copyright (C) 2005 Orielle, LLC.
-->

<!--
#=============================================================================
=
# This document provides an OWL-S specification
# for a goal of a refined threat subscription.
#
# It is used to generate a workflow composition of fuselets to achieve the
goal.
#
# There are basically two ways to perform fuselet synthesis.
# (1) The first is to generate a workflow composition of fuselets to achieve
# a goal subscription.
# (2) The second is to generate a fuselet from a specification,
# which technically can be accomplished by invoking a generic form of fuselet
# with two inputs and two outputs, and parameterized of the form
# (inTopic, inCondition, [in2...], transform, outTopic, outCondition,
[out2...])
#     A third way is to combine the two above, in a generic form
# of fuselet that acts as a fuselet and also has a component
# to invoke workflows of other fuselets in order to achieve subscription
# dependencies.
#
# Logic specification
# ===================
# The goal has a Horn clause specification of:
#    refinedThreatFuselet() {
#  IF subscribe(threatTopic, refinedThreatSelector, X).
#    }
# where uppercase words denote logical variables,
# and X denotes the resulting transform after goal solution.
#
```

```
# Service Profile
# ===============
# For brevity we include only the ServiceProcess.
#============================================================================
=
-->

<!DOCTYPE uridef[
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">

  <!ENTITY swrlx "http://www.w3.org/2003/11/swrlx">
  <!ENTITY ruleml "http://www.w3.org/2003/11/ruleml">
  <!ENTITY owlx "http://www.w3.org/2003/05/owl-xml">
  <!ENTITY swrlb "http://www.w3.org/2003/11/swrlb">

  <!ENTITY expr "http://www.daml.org/services/owl-
s/1.1/generic/Expression.owl">
  <!ENTITY swrl "http://www.w3.org/2003/11/swrl">
  <!ENTITY objList "http://www.daml.org/services/owl-
s/1.1/generic/ObjectList.owl">
  <!ENTITY time "http://www.isi.edu/~pan/damltime/time-entry.owl">
  <!ENTITY actor "http://www.daml.org/services/owl-s/1.1/ActorDefault.owl">

  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY grounding "http://www.daml.org/services/owl-s/1.1/Grounding.owl">

  <!ENTITY profileHierarchy "ProfileHierarchy.owl">
  <!ENTITY selectorHierarchy "SelectorHierarchy.owl">

  <!ENTITY fuselet_service "FuseletService.owl">
  <!ENTITY fuselet_profile "FuseletProfile.owl">
  <!ENTITY fuselet_process "FuseletProcess.owl">
  <!ENTITY fuselet_grounding "FuseletGrounding.owl">
  <!ENTITY fuselet_wsdl "FuseletGroundingWsdl.wsdl">

  <!ENTITY DEFAULT "FuseletService.owl">
  <!ENTITY THIS    "FuseletService.owl">
]>

<rdf:RDF
  xmlns:rdf = "&rdf;#"
  xmlns:rdfs  = "&rdfs;#"
  xmlns:xsd = "&xsd;#"
  xmlns:owl = "&owl;#"
  xmlns:expr = "&expr;#"
  xmlns:swrl = "&swrl;#"
  xmlns:objList =    "&objList;#"
```

```
  xmlns:swrlx = "&swrlx;#"
  xmlns:ruleml = "&ruleml;#"
  xmlns:owlx = "&owlx;#"
  xmlns:swrlb = "&swrlb;#"

  xmlns:time  = "&time;#"
  xmlns:actor = "&actor;#"

  xmlns:service = "&service;#"
  xmlns:profile = "&profile;#"
  xmlns:process = "&process;#"
  xmlns:grounding = "&grounding;#"

  xmlns:profileHierarchy = "&profileHierarchy;#"
  xmlns:selectorHierarchy = "&selectorHierarchy;#"

  xmlns:fuselet_service = "&fuselet_service;#"
  xmlns:fuselet_profile = "&fuselet_profile;#"
  xmlns:fuselet_process = "&fuselet_process;#"
  xmlns:fuselet_grounding = "&fuselet_grounding;#"
  xmlns:fuselet_wsdl = "&fuselet_wsdl;#"

  xmlns       = "&DEFAULT;#"
  xml:base  = "&DEFAULT;"
 >

 <owl:Ontology rdf:about="">
   <owl:versionInfo>
     $Id: spec.xml 1.1 2005/11/14 22:33:31Z phmills Development $
   </owl:versionInfo>
   <rdfs:comment>
   This ontology represents the OWL-S service description for the
   Fuselet web service example.
   </rdfs:comment>
   <owl:imports rdf:resource="&service;" />
   <owl:imports rdf:resource="&profile;" />
   <owl:imports rdf:resource="&process;" />
   <owl:imports rdf:resource="&grounding;" />
   <owl:imports rdf:resource="&time;" />
   <owl:imports rdf:resource="&actor;" />
   <owl:imports rdf:resource="&profileHierarchy;" />
   <owl:imports rdf:resource="&selectorHierarchy;" />
   <owl:imports rdf:resource="&fuselet_profile;" />
   <owl:imports rdf:resource="&fuselet_process;" />
   <owl:imports rdf:resource="&fuselet_grounding;" />
 </owl:Ontology>

<!--
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
#:::: FuseletProcess.owl
#:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:
```

```
-->

<!--#####################################################
    # Process Model: Abstract simple process
    #####################################################-->

<process:SimpleProcess rdf:ID="RefinedThreatProcess"
name="refinedThreatFuselet">
  <process:expandsTo rdf:resource="#RefinedThreatBPEL"/>

<!--#####################################################
    # Parameters
    #####################################################-->
<!-- Generic fuselet parameters : BPEL template supports subset of
     with fuselet specification in that it
     allows only one publish and subscribe.
     To be enhanced in follow-on work.
-->

  <process:hasParameter>
    <process:Parameter rdf:ID="sphere" default="jbi">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="inTopic" default="threatTopic">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="inSelector" default="*">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="transform" default="refinedThreat">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

  <process:hasParameter>
    <process:Parameter rdf:ID="outTopic" default="threatTopic">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>
```

```
  <process:hasParameter>
    <process:Parameter rdf:ID="outCondition" default="*">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&xsd;#string</process:parameterType>
    </process:Parameter>
  </process:hasParameter>

<!--#######################################################
    # Inputs and Outputs
    #######################################################-->
  <process:hasInput>
    <process:Input rdf:ID="threatTopicIn">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&selectorHierarchy;#threatTopic</process:paramete
rType>
    </process:Input>
  </process:hasInput>

  <process:hasOutput>
    <process:Output rdf:ID="threatTopicOut">
      <process:parameterType
rdf:datatype="&xsd;#anyURI">&selectorHierarchy;#threatTopic</process:paramete
rType>
    </process:Output>
  </process:hasOutput>

<!--#######################################################
    # Conditions: An example of an empty precondition, or guard.
    #######################################################-->
  <process:hasPrecondition>
    <expr:SWRL-Condition rdf:ID="InputExists">
   <expr:expressionLanguage rdf:resource="&expr;#SWRL"/>
   <expr:expressionBody rdf:parseType="Literal">
     <swrl:AtomList rdf:about="&rdf;#nil"/>
   </expr:expressionBody>
    </expr:SWRL-Condition>
  </process:hasPrecondition>

<!--#######################################################
    # Effects: Result for each of the publish clauses.
    #######################################################-->
  <process:hasResult>
    <process:Result rdf:ID="publishClause1">
      <rdfs:comment>
             The first if-then clause of the fuselet publishing actions.
      </rdfs:comment>

      <process:inCondition>
          <expr:SWRL-Condition rdf:ID="">
            <expr:expressionBody rdf:parseType="Literal">

              <ruleml:And>
```

```
                <ruleml:And>
                  <swrlx:individualPropertyAtom  swrlx:property="subscribe">
                    <owlx:Individual owlx:name="threatTopic" />
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:individualPropertyAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="threatSelector" />
                    <ruleml:Var>X</ruleml:Var>
                  </swrlx:classAtom>
                </ruleml:And>
              </ruleml:And>

            </expr:expressionBody>
          </expr:SWRL-Condition>
      </process:inCondition>

      <process:withOutput>
        <process:OutputBinding>
          <process:toParam rdf:resource="#threatTopicOut"/>
     <!--
          <process:valueSource>
            <process:ValueOf>
              <process:theVar rdf:resource="#threatTopicOut"/>
            </process:ValueOf>
          </process:valueSource>
     -->
        </process:OutputBinding>
      </process:withOutput>

      <process:hasEffect>
        <expr:SWRL-Expression>
          <expr:expressionBody rdf:parseType="Literal">

            <ruleml:And>
              <ruleml:And>
              </ruleml:And>
            </ruleml:And>

          </expr:expressionBody>
        </expr:SWRL-Expression>
      </process:hasEffect>
    </process:Result>
  </process:hasResult>

</process:SimpleProcess>

<!-- ######################################################################## -->
<!-- # End of RDF definitions.                                              -->
<!-- ######################################################################## -->

</rdf:RDF>
```

Figure 14. Goal Specification - Refined Threat Subscription

A.4.3.2 Specify Running Fuselet Instances.

The Prova rules that specify the running fuselet instances are shown below. This instance database, which typically would be derived from the fuselet runtime environment or its monitors, are used by the theorem prover to avoid re-instantiating a fuselet if the demand for the information it provides is already provided by a running fuselet.

```
% Facts about fuselet instances
exists_fuselet(sensorFuselet, [exists, sensorFuselet, L]) :-
  L=java.util.ArrayList(["jbi", "none", "none", "sensor", "sensorTopic",
"*"]).
```

Figure 15. Running Fuselet Instances

A.4.3.3 Create Composed Fuselet.

The script to execute the Workflow Composition Engine to create the composed fuselet is shown below. The script is a Beanshell script for running the workflow composition engine with the above demonstration files.

```
//=============================================================================
=
// Spectrum - A Framework for
// Autonomic Fuselet Specification and Composition
//
// Demonstration Script
//
// Copyright (C) 2006, Orielle, LLC.
//=============================================================================
=
import com.orielle.spectrum.Extensions.Workflow.*;


//====
// This demonstration script
// performs workflow composition from a fuselet goal and
// a set of specifications to an inferred set of fuselet invocations.
//
// The specifications are in the form of OWL-S specifications.
// The composition engine outputs a BPEL composition that
// invokes a parameterized generic fuselet BPEL web service for each
// derived fuselet invocation.
//
// The composition engine uses ancillary files including
// (1) An OWL ontology containing a hierarchy of selector terms.
// (2) A database of existing running fuselet instances, as Prova rules.
// (3) Supporting Prova rules, e.g., append.
// (4) A BPEL composition template.
//====
```

```
var w = new WorkflowComposition ();
w.setProperties("../config/properties_workflow.xml");

//====
// Read in a set of OWL-S fuselet specifications.
//====
var specs = new String[] { "spec_threat.owl", "spec_tracker.owl",
          "spec_sensor.owl" };
w.setSpecificationsFiles (specs);

//====
// Read in the OWL-S fuselet goal.
//====
var goal = "spec_goal.owl";
w.setGoalFile (goal);

//====
// Read in an OWL ontology containing a hierarchy of selector terms.
//====
w.setOntologyFile ("selector_ontology.owl");

//====
// Read in a database of existing running fuselet instances, as Prova rules.
//====
w.setInstancesFile ("instance_database.prova");

//====
// Perform workflow composition from goal and specification to invocation
set.
//
// (1) Extract the goals and specifications into Prova rules using XSLT.
// (2) Form a combined rule set using the extracted goals,
//   specifications, instance database, and supporting rules.
// (3) Run Prova on the rules to infer fuselet invocations.
// (4) Generate BPEL fuselet invocations from the inference results.
// (5) Insert the BPEL invocations into the template.
//====
var bpel = w.infer();
if (bpel.equals("No solution")) {
   print ("No solution."); }
else { print("Solution found."); };

//====
// Get other result artifacts: extract rules in readable and formal form.
//====
var rules = w.getExtractedRules();
var readable = "";
for (i : specs) {
   readable += w.extractReadableRules(
        ProvaWrapper.fileToString(i), false) + "\n";
};
readable += w.extractReadableRules(
        ProvaWrapper.fileToString(goal), true) + "\n";
```

```
ProvaWrapper.stringToFile(rules, "outputs/rules.prova");
ProvaWrapper.stringToFile(readable, "outputs/readableRules.txt");
ProvaWrapper.stringToFile(bpel, "outputs/ComposedFuselet.bpel");
```

Figure 16. Workflow Composition - Beanshell Demonstration Script

A.4.3.4 Extracted Rules used in Synthesis

For purposes of illustration, the Prova rules for workflow synthesis extracted from the fuselet specifications, goal, and instances are shown below.

```
%================
% Supporting rules
%================
% Concatenation of list in argument 2 to the list in argument 1
% is the list in argument 3
append([],L,L). % L appended to an empty list [] is L
append([X|L1],L2,[X|L3]):- % Any list L2 appended to a list starting
          % with X is a list starting with X
   append(L1,L2,L3).

% Subscription demand of a fuselet is satisfied by publication of another.
subscribe(X, Y, Z, H) :- publish(X, Y, Z, H).

%=====================
% Fuselet specification
%=====================
publish( threatTopic, S, threat( X), [P | H]) :-
   com.orielle.spectrum.Extensions.Workflow.OntologyModel.is_a(
"refinedThreatSelector", S),
   have_fuselet( threatFuselet, Pp),
   P=java.util.ArrayList(Pp),
   subscribe( trackTopic, "trackSelector", X, HX),
   append( HX, [], H).

have_fuselet( threatFuselet, P) :-
   exists_fuselet( threatFuselet, P), !.

have_fuselet( threatFuselet, P) :-
   run_fuselet( threatFuselet, P).

run_fuselet( threatFuselet, [create,  threatFuselet, L]) :-
   L=java.util.ArrayList(
   [ "jbi", "trackTopic", "*", "threat", "threatTopic", "*" ]).

%=====================
% Fuselet specification
%=====================
publish( trackTopic, S, track( X), [P | H]) :-
   com.orielle.spectrum.Extensions.Workflow.OntologyModel.is_a(
"refinedTrackSelector", S),
```

```
   have_fuselet( trackFuselet, Pp),
   P=java.util.ArrayList(Pp),
   subscribe( sensorTopic, "sensorSelector", X, HX),
   append( HX, [], H).

publish( trackTopic, S, track2( X, Y), [P | H]) :-
   com.orielle.spectrum.Extensions.Workflow.OntologyModel.is_a(
"trackSelector", S),
   have_fuselet( trackFuselet, Pp),
   P=java.util.ArrayList(Pp),
   subscribe( sensorTopic, "sensorSelector", X, HX),
   subscribe( sensorTopic2, "sensorSelector2", Y, HY),
   append( HX, HY, H).

have_fuselet( trackFuselet, P) :-
   exists_fuselet( trackFuselet, P), !.

have_fuselet( trackFuselet, P) :-
   run_fuselet( trackFuselet, P).

run_fuselet( trackFuselet, [create,  trackFuselet, L]) :-
   L=java.util.ArrayList(
   [ "jbi", "sensorTopic", "//metadata", "track", "trackTopic", "*" ]).

%======================
% Fuselet specification
%======================
publish( sensorTopic, S, sensor( data), [P | H]) :-
   com.orielle.spectrum.Extensions.Workflow.OntologyModel.is_a(
"sensorSelector", S),
   have_fuselet( sensorFuselet, Pp),
   P=java.util.ArrayList(Pp),
   append( [], [], H).

have_fuselet( sensorFuselet, P) :-
   exists_fuselet( sensorFuselet, P), !.

have_fuselet( sensorFuselet, P) :-
   run_fuselet( sensorFuselet, P).

run_fuselet( sensorFuselet, [create,  sensorFuselet, L]) :-
   L=java.util.ArrayList(
   [ "jbi", "none", "none", "sensor", "sensorTopic", "*" ]).

% Facts about fuselet instances
exists_fuselet(sensorFuselet, [exists, sensorFuselet, L]) :-
  L=java.util.ArrayList(["jbi", "none", "none", "sensor", "sensorTopic",
"*"]).

%=====
% Goal
%=====
synthesize(Instantiations) :-
```

```
   subscribe( threatTopic, "threatSelector", Z, Instantiations),
   com.orielle.spectrum.Extensions.Workflow.ProvaWrapper.setSolution(Instanti
ations).
```

Figure 17. Extracted Rules for Fuselet Synthesis

A.4.3.5 Resulting Composed Fuselet.

The BPEL for the synthesized fuselet is shown below. The resulting composed fuselet initiates both tracking and threat fuselets, but not a sensor publisher which is recognized from a set of instance facts as already running. The WSDL for the ComposedFuselet, which is not shown, is identical to that for the generic FuseletTransform except for changing the name of the program and partnerlinks.

```
<!-- FuseletTransform BPEL Process [Generated by the Oracle BPEL Designer]  -
->
<process name="ComposedFuselet"
targetNamespace="http://com.orielle.bpel.fuselets" suppressJoinFailure="yes"
xmlns:tns="http://com.orielle.bpel.fuselets"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
xmlns:jbi="http://com.orielle.bpel.publishSubscribe"
xmlns:nsxml0="http://schemas.xmlsoap.org/ws/2003/03/addressing"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
   <bpelx:exec import="java.util.*"/>
   <bpelx:exec import="java.lang.*"/>
   <bpelx:exec import="java.rmi.RemoteException"/>
   <bpelx:exec import="org.w3c.dom.Element"/>
   <bpelx:exec import="com.orielle.saffire.Extensions.JBI.*"/>
   <bpelx:exec import="com.orielle.saffire.Extensions.XmlSpaces.*"/>
   <bpelx:exec import="org.infospherics.commonAPI.impl.*"/>
   <!-- ================================================================= -->
   <!-- PARTNERLINKS                                                      -->
   <!-- List of services participating in this BPEL process               -->
   <!-- ================================================================= -->
   <partnerLinks>
     <!--
        The 'client' role represents the requester of this service. It is
        used for callback. The location and correlation information
associated
        with the client role are automatically set using WS-Addressing.
        -->
     <partnerLink name="client" partnerLinkType="tns:ComposedFuselet"
myRole="FuseletTransformProvider" partnerRole="FuseletTransformRequester"/>
     <partnerLink name="subscribeService"
partnerLinkType="jbi:PublishSubscribe" partnerRole="PublishSubscribeProvider"
myRole="PublishSubscribeRequester"/>
     <partnerLink name="publishService"
partnerLinkType="jbi:PublishSubscribe" partnerRole="PublishSubscribeProvider"
myRole="PublishSubscribeRequester"/>
```

```
     <!-- NEW WORKFLOW -->
     <partnerLink name="fuseletService"
partnerLinkType="tns:FuseletTransform" partnerRole="FuseletTransformProvider"
myRole="FuseletTransformRequester"/>
   </partnerLinks>
   <!-- ================================================================= -->
   <!-- VARIABLES                                                         -->
   <!-- List of messages and XML documents used within this BPEL process  -->
   <!-- ================================================================= -->
   <variables>
     <!-- Reference to the message passed as input during initiation -->
     <variable name="params" messageType="tns:inputParametersMessage"/>
     <!-- Reference to the message that will be sent back to the
             requester during callback
             -->
     <variable name="instanceId" messageType="tns:instanceIdMessage"/>
     <variable name="inSessionId" messageType="tns:sessionIdMessage"/>
     <variable name="isPaused" type="xsd:boolean"/>
     <variable name="subscriptionRequest"
messageType="jbi:subscriptionMessage"/>
     <variable name="inMessage" messageType="jbi:messageEnvelope"/>
     <variable name="publisherCorrelationId"
messageType="jbi:sessionIdMessage"/>
     <variable name="publishRequest" messageType="jbi:subscriptionMessage"/>
     <variable name="outMessage" messageType="jbi:messageEnvelope"/>
     <variable name="subscriberCorrelationId"
messageType="jbi:sessionIdMessage"/>
     <variable name="status" messageType="jbi:statusMessage"/>
   </variables>
   <!-- ================================================================= -->
   <!-- ORCHESTRATION LOGIC                                               -->
   <!-- Set of activities coordinating the flow of messages across the    -->
   <!-- services integrated within this business process                  -->
   <!-- ================================================================= -->
   <sequence name="main">
     <!-- Receive input from requestor.
             Note: This maps to operation defined in FuseletTransform.wsdl
             -->
     <receive name="receiveInput" partnerLink="client"
portType="tns:FuseletTransform" operation="initiate" variable="params"
createInstance="yes"/>
     <!-- Asynchronous callback to the requester.
             Note: the callback location and correlation id is transparently
handled
             using WS-addressing.
             -->
     <assign name="setInstanceId">
       <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-1"/></bpeld:annotation><copy>
         <from expression="ora:getInstanceId()"></from>
         <to variable="instanceId" part="payload"
query="/tns:InstanceId/tns:instanceId"/>
```

```
        </copy>
        <copy>
          <from expression="false()"></from>
          <to variable="isPaused"/>
        </copy>
      </assign>
      <reply name="reply-1" partnerLink="client"
portType="tns:FuseletTransform" operation="initiate" variable="instanceId"/>

      <!-- INSERTED WORKFLOW -->
      <sequence name="ComposedProcess">
        <flow name="ComposedProcess">

      <sequence>
        <scope name="FuseletInvocation">
        <variables>
        <variable name="fparams" messageType="tns:inputParametersMessage"/>
        <variable name="uid" messageType="tns:instanceIdMessage"/>
        </variables>
        <sequence>
        <assign name="SetParameters">
          <copy>
            <from>
          <Parameters xmlns="http://com.orielle.bpel.fuselets">
            <sphere>jbi</sphere>
                        <inTopic>trackTopic</inTopic>
                    <inSelector>*</inSelector>
            <transform>threat</transform>
            <outTopic>threatTopic</outTopic>
            <outCondition>*</outCondition>
          </Parameters>
             </from>
             <to variable="fparams" part="payload"/>
          </copy>
        </assign>
        <invoke partnerLink="fuseletService" portType="tns:FuseletTransform"
operation="initiate" inputVariable="fparams" outputVariable="uid"/>
        </sequence>
        </scope>
      </sequence>

      <sequence>
        <scope name="FuseletInvocation">
        <variables>
        <variable name="fparams" messageType="tns:inputParametersMessage"/>
        <variable name="uid" messageType="tns:instanceIdMessage"/>
        </variables>
        <sequence>
        <assign name="SetParameters">
          <copy>
            <from>
          <Parameters xmlns="http://com.orielle.bpel.fuselets">
            <sphere>jbi</sphere>
```

```
                          <inTopic>sensorTopic</inTopic>
                        <inSelector>//metadata</inSelector>
            <transform>track</transform>
            <outTopic>trackTopic</outTopic>
            <outCondition>*</outCondition>
          </Parameters>
             </from>
             <to variable="fparams" part="payload"/>
          </copy>
        </assign>
        <invoke partnerLink="fuseletService" portType="tns:FuseletTransform"
operation="initiate" inputVariable="fparams" outputVariable="uid"/>
       </sequence>
       </scope>
     </sequence>


       </flow>
     </sequence>
     <sequence name="TransformProcess">
       <assign name="setSubscriptionRequest">
         <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-2"/></bpeld:annotation><copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:sphere"></from>
            <to variable="publishRequest" part="sphere" query="/jbi:Sphere"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:sphere"></from>
            <to variable="subscriptionRequest" part="sphere"
query="/jbi:Sphere"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:inTopic"></from>
            <to variable="subscriptionRequest" part="topic"
query="/jbi:Topic"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:inSelector"></from>
            <to variable="subscriptionRequest" part="selector"
query="/jbi:Selector"/>
          </copy>
          <copy>
            <from variable="params" part="payload"
query="/tns:Parameters/tns:outTopic"></from>
            <to variable="publishRequest" part="topic" query="/jbi:Topic"/>
          </copy>
          <copy>
```

```
            <from variable="params" part="payload"
query="/tns:Parameters/tns:inSelector"></from>
            <to variable="publishRequest" part="selector"
query="/jbi:Selector"/>
          </copy>
        </assign>
        <switch name="switch-2">
          <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="switch-2"/></bpeld:annotation><case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:outTopic')) != 'none'">
            <invoke name="invoke-2" partnerLink="publishService"
portType="jbi:PublishSubscribe" operation="createPublisher"
inputVariable="publishRequest" outputVariable="publisherCorrelationId"/>
          </case>
          <otherwise>
            <empty name="empty-3"/>
          </otherwise>
        </switch>
        <switch name="switch-3">
          <case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:inTopic')) != 'none'">
            <invoke name="invoke-1" partnerLink="subscribeService"
portType="jbi:PublishSubscribe" operation="createSubscriber"
inputVariable="subscriptionRequest"
outputVariable="subscriberCorrelationId"/>
          </case>
          <otherwise>
            <empty name="empty-2"/>
          </otherwise>
        </switch>
        <scope name="InteractionHandler"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/">
          <eventHandlers>
            <onMessage partnerLink="client" portType="tns:FuseletTransform"
operation="stop" variable="inSessionId">
              <terminate name="terminate-1"/>
            </onMessage>
            <onMessage partnerLink="client" portType="tns:FuseletTransform"
operation="resume" variable="inSessionId">
              <empty/>
            </onMessage>
            <onMessage partnerLink="client" portType="tns:FuseletTransform"
operation="pause" variable="inSessionId">
              <assign name="setIsPaused">
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-4"/></bpeld:annotation><copy>
                  <from expression="true()"></from>
                  <to variable="isPaused"/>
                </copy>
```

```
              </assign>
            </onMessage>
          </eventHandlers>
          <while name="TransformLoop" condition="true()">
            <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="while-1"/></bpeld:annotation><sequence>
              <switch name="switch-4">
                <case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:inTopic')) != 'none'">
                  <sequence>
                    <invoke name="invoke-1" partnerLink="subscribeService"
portType="jbi:PublishSubscribe" operation="getMessage"
inputVariable="subscriberCorrelationId" outputVariable="inMessage"/>
                  </sequence>
                </case>
                <otherwise>
                </otherwise>
              </switch>
              <switch name="switch-1">
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="switch-1"/></bpeld:annotation><case
condition="boolean(bpws:getVariableData('isPaused'))">
                  <receive createInstance="no" name="receive-2"
partnerLink="client" portType="tns:FuseletTransform" operation="resume"
variable="inSessionId"/>
                </case>
                <otherwise>
                  <empty name="empty-1"/>
                </otherwise>
              </switch>
              <assign name="copyMessage">
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-2"/></bpeld:annotation><copy>
                  <from variable="inMessage"></from>
                  <to variable="outMessage"/>
                </copy>
              </assign>
              <bpelx:exec
xmlns:bpelx="http://schemas.oracle.com/bpel/extension" language="java"
version="1.4" name="transform">
                <![CDATA[
//============================================================================
=
// Java code snippet goes here ... perform transform
//============================================================================
=
   try {

   // Retrieve variables from scope
```

```
   Element ssn = (Element) getVariableData("publisherCorrelationId",
"correlationId", "/SessionId");
   String correlationId = ssn.getNodeValue();
   ssn = (Element)
getVariableData("params","payload","/tns:Parameters/tns:inTopic");
   String inTopic = ssn.getNodeValue();
   ssn = (Element)
getVariableData("params","payload","/tns:Parameters/tns:outTopic");
   String outTopic = ssn.getNodeValue();
   ssn = (Element)
getVariableData("params","payload","/tns:Parameters/tns:transform");
   String transform = ssn.getNodeValue();
   ssn = (Element) getVariableData("inMessage", "metadata", "/Metadata");
   String metadata = ssn.getNodeValue();
   ssn = (Element) getVariableData("inMessage", "payload", "/Payload");
   String content = ssn.getNodeValue();

   // Sleep for a while
   Thread.sleep(1000);

   // Perform transform
   //====
   // if (! inTopic.equals("none")) { };
   //====
   if (transform.equals("sensor")) {
     System.out.println("Sensor output generated");
     metadata = "<metadata><sensor>ewr</sensor></metadata>";
     content = "<sensorReturn>0 1 0</sensorReturn>";
   } else if (transform.equals("track")) {
     System.out.println("Track updated");
     metadata = "<metadata><track>ewr</track></metadata>";
     content = "<track>1 1 1</track>";
   } else if (transform.equals("threat")) {
     System.out.println("Threat issued");
     metadata = "<metadata><threat>ewr</threat></metadata>";
     content = "<track>1 1 1</track>";
   };
   //====
   // if (! outTopic.equals("none")) { };
   //====

   // Set result variables
   setVariableData("outMessage", "metadata", "/Metadata", metadata);
   setVariableData("outMessage", "payload", "/Payload", content);
   setVariableData("outMessage", "correlationId", "/SessionId",
correlationId);

   } catch (Exception e) { addAuditTrailEntry("Transform error", e); };
//=========================================================================
=
               ]]>
             </bpelx:exec>
             <assign name="setPubCorrelationId">
```

```
                <bpeld:annotation
xmlns:bpeld="http://schemas.oracle.com/bpel/eclipse/designer"><bpeld:meta
map.label="assign-1"/></bpeld:annotation><copy>
                  <from variable="publisherCorrelationId"
part="correlationId" query="/jbi:SessionId"></from>
                  <to variable="outMessage" part="correlationId"
query="/jbi:SessionId"/>
                </copy>
              </assign>
              <switch name="switch-5">
                <case
condition="string(bpws:getVariableData('params','payload','/tns:Parameters/tn
s:outTopic')) != 'none'">
                  <invoke name="invoke-3" partnerLink="publishService"
portType="jbi:PublishSubscribe" operation="sendMessage"
inputVariable="outMessage" outputVariable="status"/>
                </case>
                <otherwise>
                </otherwise>
              </switch>
            </sequence>
          </while>
        </scope>
      </sequence>
   </sequence>
</process>
```

Figure 18. Synthesized Fuselet Workflow